# CONCURRENT COMPILING

## Volume II - The Parallel Execution of FORTRAN DO Loops

Leslie Lamport
David Presberg

Applied Data Research, Inc.

UNCLASSIFIED
Security Classification

**DOCUMENT CONTROL DATA - R & D**

*(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)*

| | |
|---|---|
| 1 ORIGINATING ACTIVITY (Corporate author)<br>Applied Data Research, Inc.,<br>Lakeside Office Park,<br>Wakefield, Massacusetts 01880 | 2a. REPORT SECURITY CLASSIFICATION<br>UNCLASSIFIED<br>2b. GROUP<br>N/A |

3 REPORT TITLE
CONCURRENT COMPILING
Volume II - The Parallel Execution of FORTRAN DO Loops

4 DESCRIPTIVE NOTES (Type of report and inclusive dates)
Final Report

5 AUTHOR(S) (First name, middle initial, last name)
Leslie Lamport
David Presberg

| | | |
|---|---|---|
| 6 REPORT DATE<br>March 1972 | 7a TOTAL NO OF PAGES<br>156 | 7b. NO OF REFS<br>6 |
| 8a CONTRACT OR GRANT NO<br>F30602-71-C-0115<br><br>Job Order No. 45940000<br><br>d. | 9a. ORIGINATOR'S REPORT NUMBER(S)<br>CA-7112-2712<br><br>9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)<br>RADC-TR-72-64, Volume II (of two) | |

10 DISTRIBUTION STATEMENT
Approved for public release; distribution unlimited.

| | |
|---|---|
| 11 SUPPLEMENTARY NOTES<br>None | 12 SPONSORING MILITARY ACTIVITY<br>Rome Air Development Center (IRDA)<br>Griffiss Air Force Base, New York 13440 |

13 ABSTRACT

Methods are given for the analysis of FORTRAN DO loops for parallel execution on asynchronous and synchronous multi-processors. Limitations of the analytic approach are discussed. Application of the methods to the ILLIAC-IV are described. A simulation process for deriving concurrency is given in two examples for which the analytic methods are inapplicable.

DD FORM 1 NOV 65 1473

UNCLASSIFIED
Security Classification

UNCLASSIFIED
Security Classification

| 14 KEY WORDS | LINK A | | LINK B | | LINK C | |
|---|---|---|---|---|---|---|
| | ROLE | WT | ROLE | WT | ROLE | WT |
| Computer Programming | | | | | | |
| Time Sharing Multi-Processing | | | | | | |
| Concurrent Processing | | | | | | |

SAC--Griffiss AFB NY

UNCLASSIFIED
Security Classification

## FOREWORD

This document is Volume II of a two-volume Final Report by Applied Data Research, Incorporated, Lakeside Office Park, Wakefield, Massachusetts, under Contract F30602-71-C-0115, Job Order Number 45940000, for Rome Air Development Center, Griffiss Air Force Base, New York. Secondary report number is CA-7112-2712. Patricia M. Langendorf, IRDA, was the RADC Project Engineer.

This report has been reviewed by the Information Office (OI) and is releasable to the National Technical Information Service (NTIS).

This technical report has been reviewed and is approved.

Approved: Patricia M. Langendorf
PATRICIA M. LANGENDORF
Project Engineer

Approved: [illegible]
FRANZ H. DETTMER
Colonel, USAF
Chief, Intel and Recon Division

FOR THE COMMANDER: [illegible]
FRED I. DIAMOND
Acting Chief, Plans Office

## ABSTRACT

Methods are given for the analysis of FORTRAN DO loops for parallel execution on asynchronous and synchronous multi-processors. Limitations of the analytic approach are discussed. Application of the methods to the ILLIAC-IV are described. A simulation process for deriving concurrency is given in two examples for which the analytic methods are inapplicable.

TABLE OF CONTENTS

TABLE OF CONTENTS (Con't.)

## INTRODUCTION

Any program using a significant amount of computer time spends most of that time executing one or more loops. For a large class of programs, these loops can be represented as FORTRAN DO loops. We will consider methods of executing these loops on a multiprocessor computer, in which different processors independently execute different iterations of the loop at the same time.

This approach was inspired by the ILLIAC-IV, since it is the only type of parallel computation which that computer can perform [1]. However, even for a computer with independent processors, it is inherently more efficient than the usual approach of having the processors work together on a single iteration of the loop. This is because it requires much less communication among the individual processors.

The methods presented are, of course, independent of the syntax of FORTRAN. The basic feature of the FORTRAN DO loop which is used is that the range of values assumed by the index variable is known upon entry to the loop. Thus, most but not all ALGOL FOR loops can be handled.

The analysis is performed from the standpoint of a compiler for a multi-processor computer. Two types of computers are considered: those having asynchronous processors, and those like the ILLIAC-IV with completely synchronous processors. A number of restrictions are made on the loops just to simplify the exposition. In Chapter 3, we will discuss the actual limitations of the techniques. Chapter 4 describes a practical example of their use.

This approach to the parallel execution of loops appears to be very effective for a large class of programs. It has significant implications for the design of future computers and their compilers.

Chapter 5 presents, in examples, a process of loop control history simulation which can be used to display potential concurrent execution for loops which do not satisfy the requirements for the analytic methods.

# CHAPTER ONE

# ASYNCHRONOUS PROCESSORS

## I. THE GIVEN LOOP

We will consider DO loops of the following form:

```
(1)     DO α I¹ = ℓ¹, u¹
                  .
                  .
                  .
        DO α Iⁿ = ℓⁿ, uⁿ

             | loop body |

        α CONTINUE
```

where the $\ell^i$ and $u^i$ are positive integers,* and the loop body has no I/O statements, no subroutine or function calls which can modify data, and no transfer of control to any statement outside the loop. The extension to more general loops will be discussed later.

Let $\mathbb{Z}$ denote the set of all integers, and let $\mathbb{Z}^n$ denote the set of n-tuples of integers. For completeness, define $\mathbb{Z}^0 = \{0\}$.

The index set $\mathcal{J}$ of the loop (1) is defined to be the subset $\{ (i^1, \ldots, i^n) : \ell^j \leq i^j \leq u^j \}$ of $\mathbb{Z}^n$. Thus, for the loop

```
DO 7 I¹ = 1, 10
DO 7 I² = 1, 20
     .
     .
     .
```

*The use of superscripts and subscripts is in accord with the usual notation of tensor algebra.

$\mathcal{J} = \{ (x, y) : 1 \leq x \leq 10, x \leq y \leq 20 \}$ .

An execution of the loop body for an element $(p^1, \ldots, p^n)$ of $\mathcal{J}$ is the process of setting $I^1 = p^1, \ldots, I^n = p^n$ and then executing the loop body in the usual fashion, stopping when statement $\alpha$ is reached. Executing the entire loop (1) then involves the execution of the loop body for each element of $\mathcal{J}$, in the order specified by the DO statements.

This suggests that we order the elements of $\mathbb{Z}^n$ lexicographically in the usual manner, with $(2, 9, 13) < (3, -1, 10) < (3, 0, 0)$. Then for any elements P and Q of $\mathcal{J}$, the loop body is executed for P before it is executed for Q if and only if $P < Q$. Thus, the relation $<$ on $\mathbb{Z}^n$ gives the appropriate temporal ordering of $\mathcal{J}$. In the preceding example, the loop body is executed for (2, 11) before it is executed for (3, 5), since $(2, 11) < (3, 5)$.

Define addition and subtraction of elements of $\mathbb{Z}^n$ by coordinate-wise addition and subtraction, as usual. Thus, $(3, -1, 0) + (2, 2, 4) = (5, 1, 4)$. Let $\vec{0}$ denote the element $(0, 0, \ldots, 0)$. It is easy to see that for any $P, Q \in \mathbb{Z}^n$, we have $P < Q$ if and only if $Q - P > \vec{0}$.

## II. THE DO CONC STATEMENT

Our objective is to find a new temporal ordering of the executions of the loop body so that at any given time, the loop body is being executed in parallel for different elements of the index set by different processors. This new ordering must yield an algorithm which is equivalent to the one described by the original loop; i.e., one which computes the same values for all variables as the original loop.

Consider the loop

```
(2)     DO 10 I^1 = 1, 3
        DO 10 I^2 = 2, 7
        A (I^1 + 3, I^2) = 0
        10 CONTINUE
```

The loop body could be executed in parallel by three processors for the points (1, 6), (2, 5), and (3, 4) of $\mathcal{S}$. (In fact, it could be executed in parallel by 18 processors for all points in $\mathcal{S}$.)

In order to have a means of expressing parallel computation, we define the DO CONC (for CONCurrently) statement. Its form is

$$\text{DO } \alpha \text{ CONC FOR ALL } I \in \mathcal{S}$$

where $\mathcal{S}$ is a finite subset of $\mathbb{Z}$.* It has the following meaning: Let

*We remind the reader that a set is just an unordered collection of elements, so $\{1,2\} = \{2,1\} = \{1,2,1,1,2\}$. We will not bother to define a syntax for expressing sets. The usual FORTRAN DO syntax, which can only describe a restricted class of subsets of $\mathbb{Z}$, is probably the most convenient to implement.

$\mathcal{S} = \{i_1, \ldots, i_m\}$, where no two $i_j$ are equal, and assume that we have m independent, completely asynchronous processors numbered 1 through m. Then each processor is to execute the statements following the DO CONC statement, through statement $\alpha$, with processor number j setting $I = i_j$. The m processors are to run concurrently, independent of one another.

As an example, consider

```
        DO 10 CONC FOR ALL  J ε {x: 2 ≤ x ≤ 5}
        10  A(J) = J ** 2
```

This sets $A(2) = 4$, $A(3) = 9$, $A(4) = 16$ and $A(5) = 25$.

For a DO CONC to give a well-defined algorithm, certain restrictions must be made on the statements in its range. Suppose the statement

```
    9        B(J) = A(J + 1)
```

is inserted before statement 10 above. The resulting DO CONC loop does not give well-defined results. For example, the processor doing the computation for J = 3 sets B(3) to the value of A(4). But the value of A(4) it uses depends upon whether or not the processor for J = 4 has already executed statement 10. Since the processors are assumed to be asynchronous, the resulting value of B(3) is not well-defined.

We will not bother specifying the necessary restrictions on the DO CONC loop. The DO CONCs which will be written appear in loops which are equivalent to the original DO loop (1), and hence must give well-defined algorithms.

The DO CONC statement is generalized to the form

DO $\alpha$ CONC FOR ALL $(I^1, \ldots, I^k) \in \mathcal{S}$, where $\mathcal{S}$ is a subset of $\mathbb{Z}^k$. The meaning should be clear: for each element $(p^1, \ldots, p^k) \in \mathcal{S}$, we have a processor performing the calculation for $I^1 = p^1, \ldots, I^k = p^k$.

## III. REWRITING THE LOOP

Consider loop (2), with index set $\mathcal{I}$. Changing the order of execution of the loop body for the different elements of $\mathcal{I}$ obviously does not change the algorithm. The loop can, therefore, be rewritten as a single DO CONC, or in many different ways as a nested DO/DO CONC loop. Choosing one of these ways, we rewrite it as follows:

```
(3)     DO  10  $J^1 = 3, 10$

        DO  10  CONC FOR ALL $J^2 \in \{y: 2 \leq y \leq 7 \text{ and } J^1 - 3 \leq y \leq J^1 - 1\}$

        A $(J^1 - J^2 + 3, J^2) = 0$

    10  CONTINUE  .
```

The choice is arbitrary and unnatural, but instructive.

To actually construct loop (3), we first defined the one-to-one mapping $J: \mathbb{Z}^2 \to \mathbb{Z}^2$ by

$$J[\,(I^1, I^2)\,] = (I^1 + I^2, I^2) = (J^1, J^2) \ ,$$

as illustrated in Figure 1. We next defined the index set $\mathcal{J}$ to be the set $J(\mathcal{I}) = \{ J(p): p \in \mathcal{I}\}$. Then $\mathcal{J} = \{ (J^1, J^2): 3 \leq J^1 \leq 10,\ 2 \leq J^2 \leq 7$ and $J^1 - 3 \leq J^2 \leq J^1 - 1\}$, and we filled in the limits of the DO and DO CONC statements to give this index set. Finally, we rewrote the loop body in such a way that executing the body of loop (3) for the point $J(P) \in \mathcal{J}$ is equivalent to executing the body of loop (2) for the point $P \in \mathcal{I}$. In other words, $A(J^1 - J^2 + 3, J^2)$ references the same array element as $A(I^1 + 3, I^2)$ when $(J^1, J^2) = J[\,(I^1, I^2)\,]$.

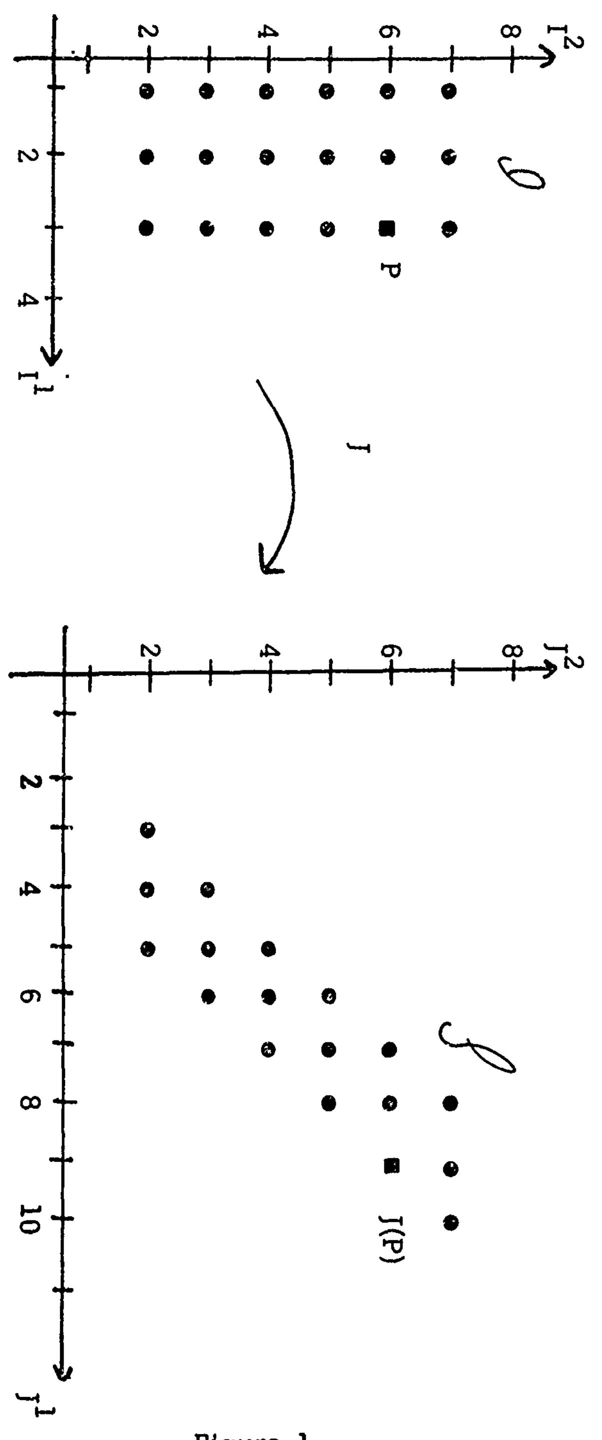


Figure 1

We can consider loop (3) to be the same as loop (2), except with a different order of execution of the body for the elements of $\mathcal{J}$. This order of execution is illustrated in Figure 2. The loop body is executed concurrently for all points in $\mathcal{J}$ lying on a straight line $J^1$ = constant. The execution for those points of $\mathcal{J}$ with $J^1 = 3$ precedes the execution for the points with $J^1 = 4$, which in turn precedes the execution for the points with $J^1 = 5$, etc.

This suggests that we define the mapping $\pi: \mathbb{Z}^2 \rightarrow \mathbb{Z}$ by

$$\pi [ (I^1, I^2) ] = J^1 = I^1 + I^2$$

Then the execution for $P \in \mathcal{J}$ precedes the execution for $Q \in \mathcal{J}$ if and only if $\pi(P) < \pi(Q)$. If $\pi(P) = \pi(Q)$, then the two executions of the loop body are concurrent.

The generalization of this rewriting procedure is straightforward. Loop (1) will be rewritten in the form

```
(4)      DO α  $J^1 = \lambda^1, u^1$
               .
               .
               .
         DO α  $J^k = \lambda^k, u^k$
         DO α  CONC FOR ALL $(J^{k+1}, \ldots, J^n) \in \mathcal{J}_{J^1, \ldots, J^k}$
           $\boxed{\text{loop body}}$
   α     CONTINUE
```

where $\mathcal{J}_{J^1, \ldots, J^k}$ is a subset of $\mathbb{Z}^{n-k}$ which may depend upon the values of $J^1, \ldots J^k$. Here, $\lambda^i$ and $u^i$ need not be integers, but may be integer

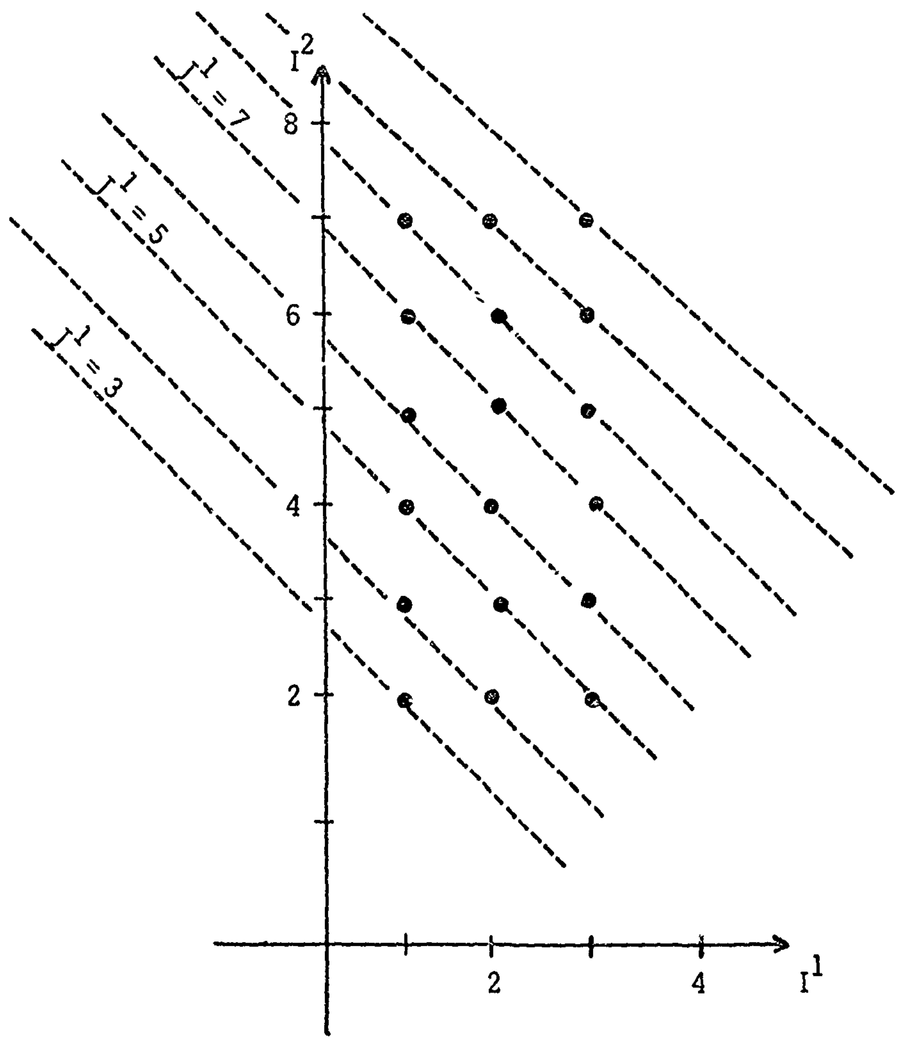


Figure 2

valued expressions whose values depend upon $J^1, \dots, J^{i-1}$.

To perform this rewriting, we will construct a one-to-one mapping $J: \mathbb{Z}^n \to \mathbb{Z}^n$ of the form

$$(5) \quad J[(I^1, \dots, I^n)] = (\sum_{j=1}^{n} a_j^1 I^j, \dots, \sum_{j=1}^{n} a_j^n I^j) = (J^1, \dots, J^n)$$

for integers $a_j^i$.* We then choose the $\lambda^i$, $u^i$ and $\mathcal{S}_{J^1, \dots, J^k}$ so that the index set $\mathcal{J}$ of the loop (4) equals $J(\mathcal{S})$, and write the body of loop (4) so that its execution for the point $J(P) \in \mathcal{J}$ is equivalent to the execution of the body of loop (1) for $P \in \mathcal{S}$.

Define the mapping $\pi: \mathbb{Z}^n \to \mathbb{Z}^k$ by

$$\pi[(I^1, \dots, I^n)] = (J^1, \dots, J^k) \quad ,$$

so $\pi(P)$ consists of the first $k$ coordinates of $J(P)$. It is then clear that for any points $J(P), J(Q) \in \mathcal{J}$, the execution of the body of loop (4) for $J(P)$ precedes the execution for $J(Q)$ if and only if $\pi(P) < \pi(Q)$. If we consider loop (4) to be a reordering of the execution of loop (1), this statement is equivalent to the following:

(E) <u>For any $P, Q \in \mathcal{S}$, the execution of the loop body for P precedes that for Q, in the new ordering of executions, if and only if $\pi(P) < \pi(Q)$.</u>

---

* $J$ is one-to-one if and only if (5) can be solved to write the $I^j$ as linear expressions in the $J^i$ with integer coefficients.

The loop body is executed concurrently for all elements of $\mathcal{J}$ lying on a set of the form $\{P: \pi(P) = \text{constant} \in \mathbb{Z}^k\}$. Since J is assumed to be a one-to-one linear mapping, these sets are parallel (n-k)-dimensional planes in $\mathbb{Z}^n$.* We thus have concurrent execution of the loop body along (n-k)-dimensional planes through the index set.

Naturally, we cannot use any arbitrary mapping J. We must find one for which loop (4) gives an algorithm equivalent to that of loop (1). This is the goal of the following analysis.

Observe that rewriting loop (1) so all executions are concurrent; i.e., with a

DO $\alpha$ CONC FOR ALL $(I^1, \ldots, I^n) \in \mathcal{J}$

statement, involves setting J equal to the identity mapping, $k = 0$, and $\pi: \mathbb{Z}^n \rightarrow \mathbb{Z}^0$ the mapping defined by $\pi(P) = 0$ for all $P \in \mathbb{Z}^n$.

*We consider $\mathbb{Z}^n$ to be a subset of ordinary Euclidean n-space, as we did in drawing Figures 2 and 3.

## IV. THE BASIC RULE

We first introduce some terminology to aid the discussion. Consider the variable VAR defined by the statement

```
DIMENSION    VAR (10, 20) .
```

The range of VAR is the set $\mathcal{R}_{VAR} = \{(x,y): 1 \leq x \leq 10,\ 1 \leq y \leq 20\}$, which is a subset of $\mathbb{Z}^2$. Thus, $\mathcal{R}_{VAR}$ is the set of all $(x,y) \in \mathbb{Z}^2$ such that VAR (x,y) is defined.*

An occurrence of VAR is any appearance of it in the loop body. If the occurrence appears as

```
VAR (-,-) = ...,
```

then it is called a generation; otherwise it is called a use. I.e., generations modify the values of elements of the array VAR, and uses do not.

Let f denote an occurrence of VAR in loop (1) of the form VAR $(I^1 + 3, I^3)$, and assume $n = 3$. During execution of the loop body for the element $(3, 4, 5) \in \mathcal{S}$, this occurrence becomes VAR (6, 5). We say that f references the point $(6, 5) \in \mathcal{R}_{VAR}$ for (3, 4, 5).

This defines an occurrence mapping $T_f : \mathcal{S} \to \mathcal{R}_{VAR}$ by

---

*For a scalar variable x, we set $\mathcal{R}_x = \mathbb{Z}^0$.

letting $T_f(P)$ be the point of $\mathcal{R}_{VAR}$ referenced by $f$ for $P \in \mathcal{I}$. In this case, $T_f$ is given by

$$T_f[(p^1, p^2, p^3)] = (p^1 + 3, p^3) .$$

We will assume that all variable occurrences only have the loop variables $I^1, \ldots, I^n$ and integer constants in their subscript expressions. Then for any variable occurrence $g$, the occurrence mapping $T_g : \mathbb{Z}^n \to \mathbb{Z}^m$ is well-defined, where $m$ is the dimension (number of subscript positions) of the variable.

Now consider the loop

$$\begin{array}{lll}
(6) & & \text{DO } 23 \quad I^1 = 2, 10 \\
& & \text{DO } 23 \quad I^2 = 3, 17 \\
& 21 & \underset{\textcircled{a1}}{A(I^1, I^2)} = \underset{\textcircled{c1}}{C(I^1)} \\
& 22 & \underset{\textcircled{b1}}{B(I^1, I^2)} = \underset{\textcircled{a2}}{A(I^1 - 1, I^2 + 1)} + \underset{\textcircled{b2}}{B(I^1, I^2)} \\
& 23 & \text{CONTINUE.}
\end{array}$$

We have introduced the convention of writing the name of an occurrence in a circle beneath it. For the point $(4, 7) \in \mathcal{I}$, the loop body is

```
21    A (4, 7) = C (4)
22    B (4, 7) = A (3, 8) + B (4, 7) .
```

The value A (3, 8) used in statement 22 is the one computed in statement 21 during execution of the loop body for the point (3,8). To ensure that the execution for (4,7) computes the right value when we change the order of executions of the body, we must only require that it be preceded by the execution for (3,8). By statement E above, this means that $\pi$ must satisfy $\pi[(3, 8)] < \pi[(4, 7)]$.

In general, let VAR be any variable. If a generation and a use of VAR both reference the same element in the range of VAR during execution of the loop, then the order of the references must be preserved. In other words, if f is a generation and g is a use of VAR, and $T_f(P) = T_g(Q)$ for some points P, Q $\in \mathcal{J}$, then:

(i) If $P < Q$, we must have $\pi(P) < \pi(Q)$,

(ii) If $Q < P$, we must have $\pi(Q) < \pi(P)$.

In the above example, $T_{a_1}[(3,8)] = T_{a_2}[(4,7)] = (3,8)$ and $(3,8) < (4,7)$, so we must have $\pi[(3,8)] < \pi[(4,7)]$. Note that if $P = Q$, then the order of execution of the references will automatically be preserved, since they happen during a single execution of the loop body. Thus, the fact that $T_{b_1}[(4,7)] = T_{b_2}[(4,7)]$ does not place any restriction on our choice of $\pi$.

The above rule should also apply to any two generations of a variable. This guarantees that the variable has the correct values after the loop is run. Together with the above rule, it also ensures that a use will always obtain the value assigned by the correct generation.

These remarks can be combined into the following basic rule:

(C1) For every variable, and every ordered pair of occurrences f, g of that variable, at least one of which is a generation: if $T_f(P) = T_g(Q)$ for $P, Q \in \mathcal{P}$ with $P < Q$, then $\pi$ must satisfy the relation $\pi(P) < \pi(Q)$.

Notice that the case $Q < P$ is obtained by interchanging f and g.

Rule C1 ensures that the new ordering of executions of the loop body preserves all relevant orderings of variable references. The orderings not necessarily preserved are those between references to different array elements, and between two uses. Changing just these orderings cannot change the value of anything computed by the loop. The assumptions we have made about the loop body, especially the assumption that it contains no exits from the loop, therefore imply that rule C1 gives a sufficient condition for loop (2) to be equivalent to loop (1).*

*For most loops, C1 is also a necessary condition.

## V. THE SETS <f, g>

The trouble with rule C1 is that it requires that we consider many pairs of points P, Q in $\mathscr{J}$. For the loop (6), there are 112 pairs of elements P, Q, $\varepsilon$ $\mathscr{J}$ with $T_{a_1}(P) = T_{a_2}(Q)$ and $P < Q$. However, $T_{a_1}(P) = T_{a_2}(Q)$ if and only if $Q = P + (1, -1)$. We would like to be able to work with the single point $(1, -1) \in \mathbb{Z}^n$, rather than all the pairs P,Q.

This suggests the following definition. For any occurrences f, g of a variable in loop (1), define the subset <f, g> of $\mathbb{Z}^n$ by

$$<f, g> = \{ X : T_f(P) = T_g(P+X) \text{ for some } P \in \mathbb{Z}^n \} .$$

Thus, for loop (6) we have $<a1, a2> = \{ (1, -1) \}$, and $<b1, b2> = \{ (0,0) \}$. Observe that <f, g> is independent of the index set $\mathscr{J}$.

We now rewrite rule C1 in terms of the sets < f, g>. First, note that $\pi(P + X) = \pi(P) + \pi(X)$, since we have assumed $\pi$ to be a linear mapping. (Recall the definition of $\pi$, and formula (5).) Also, remember that $A < A + B$ if and only if $B > \vec{0}$. Then just substituting P + X for Q in rule C1 yields:

(C1') <u>For ... generation: if $T_f(P) = T_g(P + X)$ for $P, P + X \in \mathscr{J}$ with $X > \vec{0}$, then $\pi$ must satisfy the relation $\pi(X) > \vec{0}$.</u>

Removing the clause "for P, P + X $\varepsilon$ $\mathscr{J}$ " from C1' gives a stronger condition for $\pi$ to satisfy. Doing this and using the definition of $<f,g>$ then gives the following more stringent rule:

(C2) <u>For every variable, and every ordered pair of occurrences f,g of that variable, at least one of which is a generation: for every $X \in <f,g>$ with $X > \vec{0}$, $\pi$ must satisfy $\pi(X) > \vec{0}$.</u>

Any $\pi$ satisfying C2 also satisfies C1. Hence, rule C2 gives a sufficient condition for loop (4) to be equivalent to loop (1). Moreover, C2 is independent of the index set $\mathcal{I}$.

Note here that C2 is satisfied by the zero mapping $\pi: \mathbb{Z}^n \to \mathbb{Z}^0$ if and only if it is vacuous; i.e., if and only if there are no elements $X > \vec{0}$ in any of the sets $<f,g>$ referred to in the rule. In this case, the loop body can be executed concurrently for all points in $\mathcal{I}$.

## VI. COMPUTING THE SETS $<f,g>$

We will obtain results about the existence of mappings $\pi$ satisfying rule C2. In order to do this, some restrictions must be made on the forms of the occurrences to permit a simple computation of the sets $<f,g>$. We assume that each occurrence of a variable VAR is of the form

$$(7) \qquad \mathrm{VAR}\,(I^{j_1} + m^1, \ldots, I^{j_r} + m^r) ,$$

where the $m^k$ are integer constants, and $j_1, \ldots, j_r$ are $r$ distinct integers between 1 and n. Moreover, we assume that the $j_k$ are the same for any two occurrences of VAR. Thus, if an occurrence $A\,(I^2 -1, I^1, I^4 + 1)$ appears in the loop, then the occurrence $A\,(I^2 + 1, I^1 + 6, I^4)$ may appear. However, the occurrence $A\,(I^1 -1, I^2, I^4)$ may not.

Now let $f$ be the occurrence (7) and let $g$ be the occurrence $\mathrm{VAR}\,(I^{j_1} + n^1, \ldots, I^{j_r} + n^r)$. Then

$$T_f\,[\,(p^1, \ldots, p^n)\,] = (p^{j_1} + m_1, \ldots, p^{j_r} + m_r)$$

$$T_g\,[\,(p^1, \ldots, p^n)\,] = (p^{j_1} + n_1, \ldots, p^{j_r} + n_r)$$

It is easy to see from the definition that $<f,g>$ is the set of all elements of $\mathbb{Z}^n$ whose $j_k$th coordinate is $m^k - n^k$, for $k = 1, \ldots, r$, and whose remaining $n - r$ coordinates are any integers.

As an example, suppose $n = 5$ and $f, g$ are the occurrences VAR $(I^3 + 1, I^2 + 5, I^5)$, VAR $(I^3 + 1, I^2, I^5 + 1)$. Then $<f, g> = \{(x, 5, 0, y, -1) : x, y \in \mathbb{Z}\}$ .Wewill denote this set by $(*, 5, 0, *, -1)$, so "*" means "any integer".

The index variable $I^j$ is said to be <u>missing from</u> VAR if $I^j$ is not one of the $I^{j_k}$ in (7). It is clear that $I^j$ is missing from VAR if and only if the set $<f,g>$ has an * in the $j$th coordinate, for any occurrences $f,g$ of VAR.

## VII. THE HYPERPLANE THEOREM

$I^j$ is called a missing index variable if it is missing from some generated variable in the loop; i.e., if it is missing from some variable for which a generation appears in the loop body.

The following result is an important special case of a more general result which will be given later.*

Hyperplane Concurrency Theorem: Assume that none of the index variables $I^2, \ldots, I^n$ is a missing variable. Then loop (1) can be rewritten in the form of loop (4) for $k = 1$. Moreover, the mapping $J$ used for the rewriting can be chosen to be independent of the index set $\mathcal{J}$.

Proof: First, a mapping $\pi: \mathbb{Z}^n \to \mathbb{Z}$ will be constructed which satisfies rule C2. Let $\wp$ be the set consisting of all the elements $X > \vec{0}$ of all the sets $<f,g>$ referred to in C2. We must construct $\pi$ so that $\pi(X) > 0$ for all $x \in \wp$.

Let "+" denote any positive integer, so $(+, x^2, \ldots, x^n)$ is any element of $\mathbb{Z}^n$ of the form $(x, x^2, \ldots, x^n)$ with $x > 0$. Since $I^1$

*A weaker version of this result can be found in [2].

is the only index variable which may be missing, we can write $\mathcal{P} = \{X_1, \ldots, X_n\}$ , where

$$X_r = \begin{cases} (x_r^1, \ldots, x_r^n) \\ \quad \text{or} \\ (+, x_r^2, \ldots, x_r^n) \end{cases}$$

for some integers $x_r^i$ .

The mapping $\pi$ is defined by

$$(8) \qquad \pi[\,(I^1, \ldots, I^n)\,] = a_1 I^1 + \ldots + a_n I^n$$

for non-negative integers $a_i$, to be chosen below. Since $a_1 \geqq 0$, $\pi[\,(1, x_r^2, \ldots, x_r^n)\,] > 0$ implies $\pi[\,(x, x_r^2, \ldots, x_r^n)\,] > 0$ for any integer $x > 0$ . Therefore, each $X_r$ of the form $(+, x_r^2, \ldots, x_r^n)$ can be replaced by $X_r = (1, x_r^2, \ldots, x_r^n)$ and it is sufficient to construct $\pi$ such that $\pi(X_r) > 0$ for each $r = 1, \ldots, N$. Recall that each $X_r > 0$.

Define $\mathcal{P}_j = \{X_r :\ x_r^1 = \ldots = x_r^{j-1} = 0,\ x_r^j \neq 0\}$, so $\mathcal{P}_j$ is the set of all $X_r$ whose jth coordinate is the left-most non-zero one. Then each $X_r$ is an element of some $\mathcal{P}_j$.

Now construct the $a_j$ sequentially for $j = n, n-1, \ldots, 1$ as follows. Let $a_j$ be the smallest non-negative integer such that

$$a_j x_r^j + \ldots + a_n x_r^n > 0$$

for each $X_r = (0, \ldots, 0, x_r^j, \ldots, x_r^n) \in \mathcal{P}_j$. Since $X_r > \vec{0}$ and

$x_r^j \neq 0$ implies $x_r^j > 0$ , this is possible.

Clearly, we have $\pi(X_r) > 0$ for all $X_r \in \mathcal{P}_j$. But each $X_r$ is in some $\mathcal{P}_j$, so $\pi(X_r) > 0$ for each $r = 1, \dots, n$. Thus, $\pi$ satisfies rule C2. Observe that the first non-zero $a_j$ that was chosen must equal 1, so 1 is the greatest common divisor of the $a_j$. (If all the $a_j$ are zero, then $\mathcal{P}$ must be empty, so we can let $\pi[(I^1, \dots, I^n)] = I^1$ .) A classical number theoretic calculation, described on Page 31 of [3], and reproduced in Appendix A, then gives a one-to-one linear mapping $J: \mathbb{Z}^n \to \mathbb{Z}^n$ such that

$$J[(I^1, \dots, I^n)] = (\pi[(I^1, \dots, I^n)], \dots)$$

Since the sets $<f,g>$ are independent of the index set $\mathcal{I}$, the construction of $\pi$ and $J$ given above is also independent of $\mathcal{I}$. This completes the proof. ∎

Loop (4) for $k = 1$ executes the loop body concurrently for all points in $\mathcal{I}$ lying along parallel $(n - 1)$-dimensional hyperplanes, hence the name of the theorem.

Observe that the theorem is trivially true without the restriction that $J$ be independent of $\mathcal{I}$ , because given any set $\mathcal{I}$ we can construct a $J$ for which the sets $\mathcal{S}_{J^2, \dots, J^n}$ contain at most one element, and the order of execution of the loop body is unchanged. For example , if $\mathcal{I} = \{(x,y,z): 1 \leq x \geq 10, 1 \geq y \leq 5, 1 \leq z \leq 7\}$, let $J[(x,y,z)] = (35x + 7y + z, x, y)$. Such a $J$ is clearly of no interest. However, because the mapping J provided by the theorem depends only on the loop body, it will always give real concurrent execution for a large enough index set.

Condition C2 gives a set of constraints on the mapping $\pi: \mathbb{Z}^n \to \mathbb{Z}$. The Hyperplane Theorem proves the existence of a $\pi$ satisfying those constraints. We now consider the problem of making an optimal choice of $\pi$.

It seems most reasonable to minimize the number of steps in the outer DO $J^1$ loop of loop (4). (Remember that $k = 1$.) If a sufficiently large number of processors are available, then this gives the maximum amount of concurrent computation. This means that we must minimize $\mu^1 - \lambda^1$ in loop (4). But $\lambda^1$ and $\mu^1$ are just the upper and lower bounds $\{\pi(P) : P \in \mathcal{J}\}$. Setting

$$M^i = u^i - \ell^i ,$$

it is easy to see that $u^1 - \lambda^1$ equals

$$(9) \qquad M^1 \mid a_1 \mid + \ldots + M^n \mid a_n \mid \quad ,$$

where the $a_i$ are defined by (8). Finding an optimal $\pi$ is thus reduced to the following integer programming problem: find integers $a_1, \ldots, a_n$ satisfying the constraint inequalities given by rule C2, which minimize the expression (9).

Observe that the greatest common divisor of the resulting $a_i$

must be 1. This follows because the constraints are of the form

$$x^1 a_1 + \ldots + x^n a_n > 0 \quad ,$$

so dividing the $a_i$ by their g.c.d. gives new values of $a_i$ satisfying the constraints, with a smaller value for (9). Hence, the method of [3] can be applied to find the mapping $J$.

Although the above integer programming problem is algorithmically solvable, we know of no practical method of finding a solution in the general case. However, the construction used in proving the Hyperplane Theorem should provide a good choice of $\pi$ . In fact, for most reasonable loops it actually gives the optimal solution.

## VIII. AN EXAMPLE

Now consider the following loop:

```
(10)   DO  16  $I^1 = 1, 25$
       DO  16  $I^2 = 2, 19$
       DO  16  $I^3 = 2, 29$
       $\underset{\text{f1}}{F(I^2, I^3)} = (\underset{\text{f2}}{F(I^2+1, I^3)} + \underset{\text{f3}}{F(I^2, I^3+1)}$
       x       $+ \underset{\text{f4}}{F(I^2-1, I^3)} + \underset{\text{f5}}{F(I^2, I^3-1)}) * .25$
       16      CONTINUE  .
```

It is a simplified version of a standard relaxation computation for a 20 by 30 array F, performed 25 times.

To apply the method of analysis, first perform the following calculations:

1. Compute the sets $\langle f, g \rangle$ referred to by rule C2.
2. Find all elements $X > \vec{0}$ in these sets.
3. Find the constraints on the $a_i$ implied by $\pi(X) > \vec{0}$ .

This is done in Table 1.

Next, choose $a_1, a_2, a_3$ consistent with these constraints,

| Sets | Elements > $\vec{0}$ | Constraints |
|---|---|---|
| <f1, f1> = (*, 0, 0) | (+, 0, 0) | $a_1 > 0$ |
| <f1, f2> = (*, -1, 0) | ( +, -1, 0) | $a_1 - a_2 > 0$ |
| <f2, f1> = (*, 1, 0) | ( +, 1, 0) | $a_1 + a_2 > 0$ |
|  | ( 0, 1, 0) | $a_2 > 0$ |
| <f1, f3> = (*, 0, -1) | ( +, 0, -1) | $a_1 - a_3 > 0$ |
| <f3, f1> = (*, 0, 1) | ( +, 0, 1) | $a_1 + a_3 > 0$ |
|  | ( 0, 0, 1) | $a_3 > 0$ |
| <f1, f4> = (*, 1, 0) | same as <f2, f1> |  |
| <f4, f1> = (*, -1, 0) | same as <f1, f2> |  |
| <f1, f5> = (*, 0, 1) | same as <f3, f1> |  |
| <f5, f1> = (*, 0, -1) | same as <f1, f3> |  |

Table 1

and minimizing

$$24 | a_1 | + 17 | a_2 | + 27 | a_3 | .$$

It is easy to see that the solution to this problem is $a_1 = 2$, $a_2 = 1$, $a_3 = 1$, so $\pi$ is given by

$$\pi [ (I^1, I^2, I^3) ] = 2 I^1 + I^2 + I^3 .$$

Note that this is the $\pi$ computed by the algorithm used in the proof of the Hyperplane Theorem.

Application of the algorithm described in the appendix gives the following mapping J :

$$J [ (I^1, I^2, I^3) ] = (J^1, J^2, J^3) = (2 I^1 + I^2 + I^3, I^1, I^3 ).*$$

Using this, and the inverse relation

$$(I^1, I^2, I^3) = (J^2, J^1 - 2J^2 - J^3, J^3 ) ,$$

the above loop is rewritten as follows:

```
     DO  16  J¹ = 6, 98
     DO  16  CONC FOR ALL (J², J³) ε { (x,y):
X          1 ≤ x ≤ 25, 2 ≤ y ≤ 29 and J¹ - 19 ≤ 2x + y ≤ J¹ - 2 }
     F (J¹ - 2*J² - J³, J³) = (F (J¹ - 2 * J² - J³ + 1, J³) +
X          F (J¹ - 2 * J² - J³, J³ + 1) + F (J¹ - 2 * J² - J³ - 1, J³)
```

*It is also easy to obtain this from the following fact: the mapping $J: \mathbb{Z}^n \to \mathbb{Z}^n$ defined by (5) is one-to-one if and only if the determinant of the $a_j^i$ is $\pm 1$ .

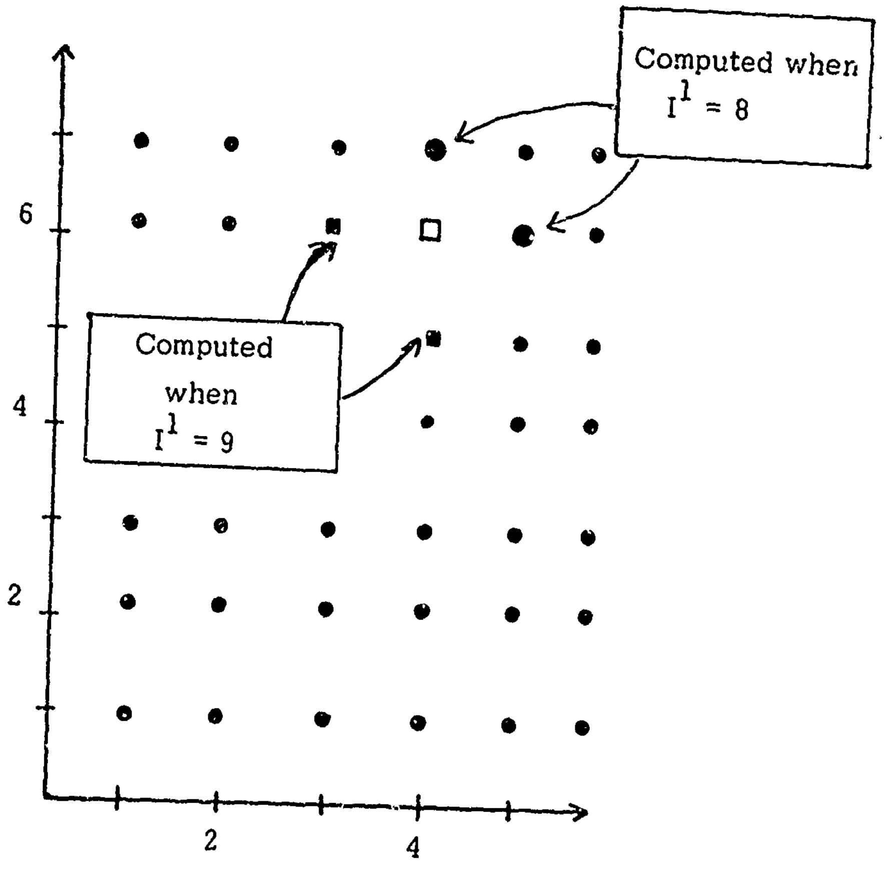
```
     X              +F (J^1 - 2 * J^2 - J^3, J^3 - 1) )* .25
 16       CONTINUE   .
```

The set expression in the DO CONC statement was obtained by writing the relations $1 \le I^1 \le 25$, $2 \le I^2 \le 19$ and $2 \le I^3 \le 29$ in terms of $J^1$, $J^2$, $J^3$ .

To understand why the rewritten loop gives the same results, consider the computation of F (4, 6) in the execution of the original loop body for the element (9, 4, 6) $\in \mathcal{I}$ . It is set equal to the average of its four neighboring array elements: F (5, 6), F (4, 7), F (3, 6), F (4, 5). The values of F (5, 6) and F (4, 7) were calculated during the execution of the loop body for (8, 5, 6) and (8, 4, 7), respectively; i.e., during the previous execution of the DO $I^1$ loop, with $I^1 = 8$. The values of F (3, 6) and F (4, 5) were calculated during the current executi f the outer DO loop, with $I^1 = 9$. This is shown in Figure 3.

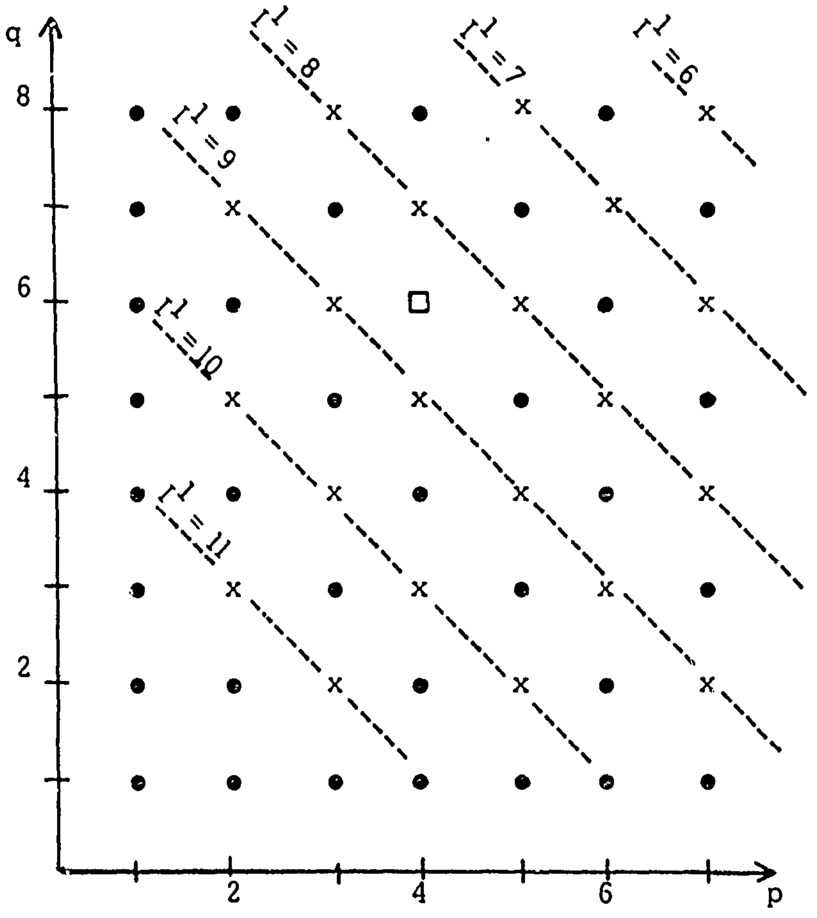
Now consider the rewritten loop. At any time during its execution, F (p,q) is being computed concurrently for up to half the elements (p,q) in the range set $\mathcal{R}_F$ of F . These computations are for different values of $I^1$. Figure 4 illustrates the execution of the DO CONC for $J^1 = 27$. The points (p,q) $\in \mathcal{R}_F$ for which F (p,q) is being computed are marked with "x"s, and the value of $I^1$ for the computation is indicated. Figure 5 shows the same thing for $J^1 = 28$.

Note how the values being used in the computation of F (4, 6)



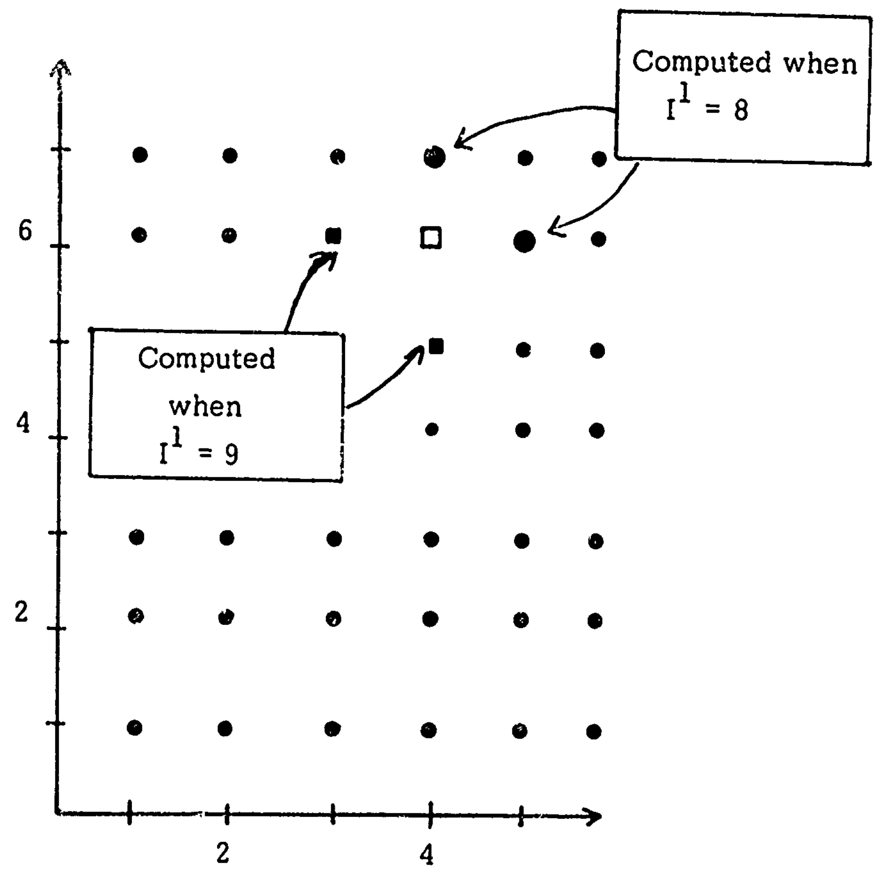
Computation of F(4, 6) for $I^1 = 9$

Figure 3



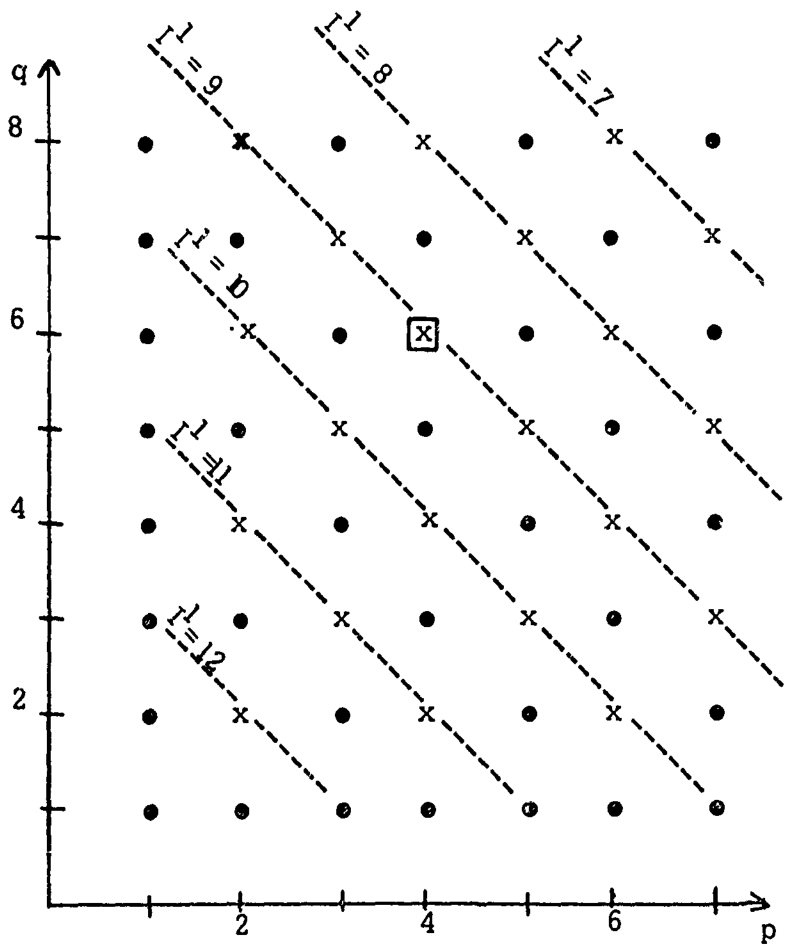
Execution for $J^1 = 27$

Figure 4



Computation of F(4, 6) for $I^1 = 9$

Figure 3



Execution for $J^1 = 28$

Figure 5

in Figure 5 were computed in Figure 4. A comparison with Figure 3 illustrates why this method of concurrent execution is equivalent to the original algorithm.

The saving in execution time achieved by the rewriting will depend upon the amount of overhead in the implementation of the DO CONC, as well as the actual number of processors available. (The sets in the DO CONC statement contain up to 252 elements.) However, the value of this approach is indicated by the fact that the number of sequential iterations has been reduced by a factor of over 135. (However, we must point out that a real program would probably include a convergence test within the outer DO $I^1$ loop, so the analysis could only be applied to the inner DO $I^2$/DO $I^3$ loop.)

## IX. THE GENERAL PLANE THEOREM

We now generalize the Hyperplane Theorem to cover the case when some of the index variables $I^2, \ldots, I^n$ are missing. Concurrent execution is then possible for the points in $\mathscr{S}$ lying along parallel planes. Each missing variable may lower the dimension of the planes by one.

Plane Concurrency Theorem: Assume that at most $k - 1$ of the index variables $I^2, \ldots, I^n$ are missing. Then loop (1) can be rewritten in the form of loop (4). Moreover, the mapping J used for the rewriting can be chosen to be independent of the index set $\mathscr{S}$.

Proof: The proof is a generalization of the proof of the Hyperplane Theorem. Let $I^{j_2}, \ldots, I^{j_k}$ be the possibly missing variables among $I^2, \ldots, I^n$. Set $j_1 = 1$, $j_{k+1} = n + 1$, and assume $j_1 < j_2 < \ldots < j_k < j_{k+1}$.

Let $\mathscr{P}$ be the set of all elements $X > 0$ of all the sets $<f,g>$ referred to by rule C2. We must construct $\pi$ so that $\pi(X) > 0$ for all $x \in \mathscr{P}$. Let $\mathscr{P}_j = \{(0, \ldots, 0, x^j, \ldots, x^n) \in \mathscr{P}: x^j > 0\}$, so $\mathscr{P}_j$ is the set of all elements of $\mathscr{P}$ whose $j$th coordinate is the left-most non-zero one. Then every element of $\mathscr{P}$ is in one of the $\mathscr{P}_j$.

The mapping $\pi: \mathbb{Z}^n \rightarrow \mathbb{Z}^k$ will be constructed with

$$\pi(P) = (\pi^1(P), \ldots, \pi^k(P)),$$

where each $\pi^i: \mathbb{Z}^n \rightarrow \mathbb{Z}$ will be defined by

$$\pi^i[(I^1,\ldots,I^n)] = a_1^i I^1 + \ldots + a_n^i I^n$$

for non-negative integers $a_j^i$. Moreover, we will have $a_j^i = 0$ if $j < j_i$ or $j \geq j_{i+1}$. This implies that is $X \in P_j$ and $j > J_{i+1}$. Then $\pi^i(X) = 0$. It therefore suffices to construct $\pi^i$ so that for each j with $j_i \leq j < j_{i+1}$, and $X \in P_j$: $\pi^i(X) > 0$ – for we then have

$$\pi(X) = (0,\ldots,0,\pi^i(X),\ldots,\pi^k(X)) > 0.$$

Recall that for the sets <f,g>, an * can appear only in the $j_1,\ldots,j_k$ coordinates. Thus any element of any of the sets $P_j$ with $j_i < j < j_{i+1}$ can be represented in the form

$$(0,\ldots,0,x_r^{j_i},\ldots,x_r^{j_{i+1}-1},\ldots), \text{ or}$$

$$(0,\ldots,0,+,x_r^{j_{i+1}},\ldots,x_r^{j_{i+1}-1},\ldots)$$

for a finite collection of integers $x_r^j$, $j_i \leq j < j_{i+1}$. By the same argument used in the proof of the Hyperplane Theorem, we can replace "+" by $x_r^{j_i} = 1$, and choose $a_j^i \geq 0$, $j_i \leq j < j_{i+1}$ such that

$$a_{j_i}^i x_r^{j_i} + \ldots + a_{j_{i+1}-1}^i x^{j_{i+1}-1} > 0$$

for each r. Choosing $a_j^i = 0$ for $j < j_i$ and $j > j_{i+1}$ completes the construction of the required $\pi^i$.

The construction given in Appendix A is then applied to give invertable relations of the form

$$J^{j_i} = a_{j_i}^i I^{j_i} + \ldots + a_{j_{i+1}-1}^i I^{j_{i+1}-1}$$

$$J^j = \sum_{r=j_i}^{j_{i+1}-1} b_r^j I^r \qquad \text{for } j_i < j < j_{i+1}.$$

Combining these and reording the $J^j$ gives the required mapping J. ∎

As in the hyperplane case, to get an optimal solution we want to minimize the number of iterations of the outer DO loops. This means minimizing $(\mu^1 - \lambda^1 + 1) \ldots (\mu^k - \lambda^k + 1)$. Using the notation of (5), it is easy to verify that this number is equal to

(11) $(M^1 | a_1^1 | + \ldots + M^n | a_n^1 | + 1) \ldots (M^1 | a_1^k | + \ldots + M^n | a_n^k | + 1)$,

where $M^i = \mu^i - \ell^i$.

Finding the $a_j^i$ is now an integer programming problem. Note that a solution with $a_1^i = \ldots = a_n^i = 0$ for some i gives a solution to the rewriting problem with k replaced by k-1, since that $\pi^i$ can be removed without affecting the constraint inequalities. The Plane Concurrency Theorem proves the existence of a $\pi: \mathbb{Z}^n \to \mathbb{Z}^k$ satisfying C2, for a particular value of k. It may be possible to find such a $\pi$ for a smaller k.

For completeness, we will state a sufficient condition for the loop body to be concurrently executable for all points in $\mathcal{L}$. This is the case when the zero mapping ($\pi(P) = 0$ for all $P \in \mathbb{Z}^n$) satisfies C2. Since $<g,f> = \{ -X : X \in <f,g> \}$, it is clear that this is true if and only if all the sets $<f, g>$ are equal to $\{0\}$. Finally, the rules for computing the sets $<f,g>$ give the following rather obvious result:

<u>If none of the index variables are missing, and for any generated variable, all occurrences of that variable are identical, then loop (1) can be rewritten as a</u>

DO $\alpha$ CONC FOR ALL $(I^1, \ldots, I^n) \in \mathcal{J}$

<u>loop.</u>

The hypothesis means that in the expression (7) for any generated variable VAR, $r = n$ and the $m^i$ are the same for all occurrences of VAR.

# CHAPTER TWO

# SYNCHRONOUS PROCESSORS

## 1. The DO SIM Statement

We now consider the case of completely synchronous processors, the primary example being the ILLIAC-IV. To accomodate it, let us introduce the DO SIM (for SIMultaneously) statement, having the following form:

DO $\alpha$ SIM FOR ALL $(I^1,\ldots,I^k) \in \mathcal{J}$,

where $\mathcal{J}$ is a subset of $\mathbb{Z}^k$. Its meaning is similar to that of the DO CONC statement, except that the computation is performed synchronously by the individual processors. Each point of $\mathcal{J}$ is assigned to a separate processor, and each statement in the range of the DO SIM is, in turn, simultaneously executed by all the processors. An assignment statement is executed by first computing the right-hand side, then simultaneously performing the assignment.

As an example, consider

```
      DO 15 SIM FOR ALL I ε { x : 2 ≤ x ≤ 10 }
14    A(I) = A(I-1) + B(I)
15    B(I) = A(I) ** 2
```

The right-hand side of statement 14 is executed for all I before the assignment of A(I) is made, and before statement 15 is executed. Therefore, if initially A(4) = 5 and B(5) = 2, then executing the loop sets A(5) = 7 and B(5) = 49.

Because of the simultaneity of execution of the body for the various points of $\mathcal{J}$, we cannot allow any conditional transfer of control in the loop body which depends upon the index variables. E. g., the statement

```
IF (A(I)) 3, 4, 5
```

may not appear in a "DO SIM FOR ALL I" loop.

For simplicity, assume that there is no transfer of control within the body of the loop, so the statements are always executed sequentially in the order in which they appear.

We will allow conditional assignment statements such as

IF (A(I).EQ. 0) B(I) = 3.

They are easily implemented on the ILLIAC-IV because of its ability to turn off individual processors.

The only other restriction to be made on the body of a DO SIM loop is that a generation may not reference the same array element for two different points in its index set $\mathcal{S}$. I.e., an assignment statement may not have two different processors simultaneously storing values into a single memory location. We do allow them to simultaneously load a value from a single memory location, so this restriction is not made for uses of a variable. *

* Simultaneous loads from a single memory location are implemented in the ILLIAC-IV by the ability of the central control unit to broadcast a value to all processors.

## II. Rewriting the Loop

Now consider the problem of rewriting the given loop (1) in the form

```
(12)  DO α J^1 = λ^1, μ^1
          :
      DO α J^k = λ^k, μ^k
      DO α SIM FOR ALL (J^{k+1},...,J^n) ε 𝒥_{J^1,...,J^k}
                    [loop body]
      α CONTINUE
```

This is the same as loop (4), except the DO CONC is replaced by a DO SIM. We assume that the body of loop (1) contains no control transfering statements - i.e., no GO TO or numerical IF statements.

Define the mappings J and π as before. Any DO CONC statement can be executed as a DO SIM, since it must give the same result if the asynchronous processors happen to be synchronized. Thus, the rewriting could be done just as before by finding a π which satisfies C2. However, the synchrony of the computation will allow us to weaken the condition C2.

Recall that rule C1 was made so that the rewriting will preserve the order in which two different references are made to the same array element. For references made during two different executions of the loop body, the asynchrony of the processors requires that the order of those executions be preserved. However, with synchronous processors, we can allow the two loop body executions to be done simultaneously if the references will then be made in the correct order. The order of these two references is determined by the positions within the loop body of the occurrences which do the referencing.

For two occurrences f and g, let $f << g$ denote that the execution of f precedes the execution of g within the loop body. This means that either the statement containing f precedes the statement containing g, or else that f is a use and g a generation in the same statement. The above observation allows us to change rule C1 to the following weaker condition on $\pi$:

<u>For ... generation: if $T_f(P) = T_q(Q)$</u>

<u>for P, Q $\epsilon$ with $P < Q$, then we must have either</u>

<u>(i) $\pi(P) < \pi(Q)$, or</u>

<u>(ii) $\pi(P) = \pi(Q)$ and $f << g$</u>.

In this rule, either (i) or (ii) is sufficient to insure that occurrence f references $T_f(P)$ for the point $P \in \mathcal{J}$ before g references the same element for $Q \in \mathcal{J}$.

The conditions can be rewritten in the following equivalent form:

(i) $\pi(P) \leq \pi(Q)$ <u>and</u> (ii) if $\pi(P) = \pi(2)$ then $f \ll g$.

In the same way c2 was obtained from cl, the above rule gives the following:

(S1) <u>For every variable and every ordered pair of occurrences f, g of that variable, at least one of which is a generation: for every $X \in <f, g>$ with $X > \vec{0}$, we must have</u>:

(i) $\pi(X) \geq \vec{0}$, and

(ii) if $\pi(X) = \vec{0}$, than $f << g$.

If $\pi$ satisfies rule S1, then it satisfies the preceding rule, so the rewritten loop (12) is equivalent to the original loop (1).

We have been assuming that in rewriting the loop body, the order of execution of the occurrences was not changed. I.e., the $I^j$ were replaced by expressions involving $J^1, \ldots, J^n$, but nothing else was done to the loop body. Now let us consider changing the order of execution of the occurrences.*

That is, we may change the position of occurrences within the loop body. For example, we may reorder the statements.

Let $f<<g$ mean that f is executed before g in the rewritten loop body (12). Then rule S1 guarantees that the correct temporal ordering of references is maintained when the references were made in the original loop during *different* executions of the loop body. Having changed the positions of occurrences in rewriting the loop body, we now have to make sure that any two references to the same array element made during a *single* execution of the loop body are still made in the correct order. The following analogue of rule C1 handles this:

For ... generation: if $T_f(P) = T_g(P)$ for some $P \in \mathcal{S}'$, and f precedes g in the original loop body. Then $f << g$.

* Remember that there was no point in doing this before, since it couldn't help for asynchronous processors.

Rewriting this in terms of the sets $<f,g>$ gives the following rule.

<u>(S2) For every variable, and every ordered pair of occurrences f, g of that variable, at least one of which is a generation: if $\vec{0} \in <f, g>$ and f precedes g in the original loop body, then $f << g$</u>.

Rules S1 and S2 guarantee that the rewritten loop (12) is equivalent to the original loop (1). Note that rule S2 does not involve $\pi$.

## III. The Coordinate Method

We could now try to solve the following problem: find a rewriting of the loop body (and the resulting << relation between occurrences) and a mapping $\pi$ which satisfy rules S1 and S2, and which minimize the expression (11). This would give a rewriting of the loop which is optimal in the sense that the outer DO $J^1$ /.../ DO $J^k$ loop has the fewest iterations. However, the optimality of such a rewriting is illusory, for reasons which we will now discuss.

The ILLIAC-IV has 64 processors. The feasibility of a machine with so many processors is achieved by having all processors operate synchronously with a single control unit, and by allowing each processor to access only its own separate portion of memory. If processor 12 wants to load a data word contained in processor 5's part of memory, then the following sequence of instructions is executed simultaneously by each processor number i, for i = 0 to 63:

(1) load

(2) transmit data word to processor i + 7 (mod 64).

This means that the method of storing arrays must depend upon how they are to be accessed. For example, consider the occurrence $F(J^1 - 2 * J^2 - J^3, J^3)$ inside the DO CONC FOR ALL $(J^2, J^3)$, which we generated before with the Hyperplane Theorem. It necessitates a complicated, space-wasting format for storing the array F. The array would probably have to be reformated before and after execution of the outer DO $J^1$ loop. *

It appears that the best results are obtained by choosing a mapping J which gives a loop with simple subscripting and a reasonable amount of simultaneous computation. An obvious way of choosing such a J is to let $J^1,\ldots,J^n$ be a permutation of the original index variables $I^1,\ldots,I^n$. More precisely, the mapping $\pi$: $\mathbb{Z}^n \to \mathbb{Z}^k$ is taken to be a coordinate projection - that is, a mapping for which $\pi[(a^1,\ldots,a^n)]$ is obtained by deleting n-k coordinates from $(a^1,\ldots,a^n)$.

For example, for n = 5 we may want to rewrite loop (1) as

```
DO α I³ = ℓ³, u³
DO α I⁴ = ℓ⁴, u⁴
DO α SIM FOR ALL (I¹, I², I⁵) ε {(x, y, z):
      ℓ¹ ≤ x ≤ u¹, ℓ² ≤ y ≤ u², and ℓ⁵ ≤ z ≤ u⁵}
          ⋮
```

Then $\pi[(I^1, I^2, I^3, I^4, I^5)] = (I^3, I^4)$ and $J[(I^1,\ldots,I^5)] = (I^3, I^4, I^1, I^2, I^5)$. Notice that if $\pi$ is a corrdinate projection, then the sets $\mathscr{S}_{J^1,\ldots,J^k}$ of loop (11) are easy to compute.

The coordinate method consists of first choosing a coordinate projection $\pi$, and then trying to find a rewriting of the loop body for which S1 and S2 are satisfied. Since rewriting the loop body makes no difference

* A precise statement of the rules relating storage allocation and DO SIMs is contained in [4].

to condition (i) of S1, we must first require that it be satisfied for all relevant occurrences f, g. Next, we apply S1 and S2 to get certain ordering relations << between occurrences. We must then decide if it is possible to rewrite the loop body so that these relations are satisfied.

In order to make this decision, we need a trivial observation: a use in an assignment statement must precede the generation in that statement. This observation will be given the status of a rule.

(S3) For any use f and generation g in a single statement, we must have f << g.

Now we add the relations << given by S3 to those obtained from S1 and S2. Next, we add all relations implied by transitivity. I. e., whenever f << g and g << h, we must add the relation f << h.* If the resulting ordering relations are consistent - that is, if we do not have f << f for any occurrence f - then the loop body can be rewritten to satisfy the ordering relations. We will describe the method of rewriting the loop body by an example.

* An efficient algorithm for doing this is given by [5].

## IV. An Examp'

Consider the following simple loop:

DO 24 $I^1 = 2, 50$

DO 24 $I^2 = 1, 5$

21 $\underset{\textcircled{a1}}{A(I^1, I^2)} = \underset{\textcircled{b1}}{B(I^1, I^2)} + \underset{\textcircled{c1}}{C(I^1)}$

22 $\underset{\textcircled{c2}}{C(I^1)} = \underset{\textcircled{b2}}{B(I^1 -1, I^2)}$

23 $\underset{\textcircled{b3}}{B(I^1, I^2)} = \underset{\textcircled{a2}}{A(I^1 + 1, I^2)} ** 2$

24 CONTINUE

We want to rewrite it as a DO $I^2$/DO SIM FOR ALL $I^1$ loop, so we apply the coordinate method with the coordinate projection $\pi$ defined by $\pi[(I^1, I^2)] = I^2$. We proceed as follows. (The calculations for steps 1-3 are shown in Table 2.)

1. Compute the relevant sets $<f, g>$ for rules S1 and S2.
2. Check that S1 (i) is not violated.
3. Find the ordering relations given by S1(ii) and S2.
4. Apply S3 to get the following relations:

| | |
|---|---|
| statement 21: | b1 << a1 |
| | c1 << a1 |

| The Sets <f, g> | Is S1 (i) Violated ? | Ordering Relations S1 (ii) | S2 |
|---|---|---|---|
| <a1, a1> = (0,0) | NO | - | - |
| <a1, a2> = (-1,0) | NO | - | - |
| <a2, a1> = (1, 0) | NO | a2<<a1 | - |
| <b3, b3> = (0,0) | NO | - | - |
| <b1, b3> = (0,0) | NO | - | b1<<b3 |
| <b3,b1> = (0,0 ) | NO | - | - |
| <b2,b3 > = (-1,0) | NO | - | - |
| <b3,b2> = (1,0) | NO | b3<<b2 | - |
| <c1,c1> = (0,0) | NO | - | - |
| <c1, c2> = (0,*) | NO | - | c1<<c2 |
| <c2, c1> = (0,*) | NO | - | - |

Table 2

| | |
|---|---|
| statement 22: | $b2 \ll c2$ |
| statement 23: | $a2 \ll b3$ |

5. Find all relations implied by transitivity:

| | |
|---|---|
| $b3 \ll c2$ | [by $b3 \ll b2$ and $b2 \ll c2$] |
| $a2 \ll b2$ | [by $a2 \ll b3$ and $b3 \ll b2$] |
| $b1 \ll b2$ | [by $b1 \ll b3$ and $b3 \ll b2$] |
| $b1 \ll c2$ | [by $b1 \ll b2$ and $b2 \ll c2$] |
| $a2 \ll c2$ | [by $a2 \ll b3$ and $b3 \ll c2$] |

6. Check that no relation of the form $f \ll f$ was found in 3 or 5.

7. Order the generations in any way which is consistent with the above relations - i.e., obeying $b3 \ll c2$. We let $a1 \ll b3 \ll c2$.

   We then write

$$\begin{array}{ll} 21 & \underset{\textcircled{a1}}{A(I^1, I^2)} = \\[1em] 23 & \underset{\textcircled{b3}}{B(I^1, I^2)} = \\[1em] 22 & \underset{\textcircled{c2}}{C(I^1)} = \end{array}$$

8. Insert the uses in positions implied by the ordering relations (recall that $a2 \ll a1$):

$$\begin{array}{ll} & \qquad\qquad \underset{\textcircled{a2}}{A(I^1 + 1, I^2)} \\[1em] 21 & \underset{\textcircled{a1}}{A(I^1, I^2)} = \underset{\textcircled{b1}}{B(I^1, I^2)} + \underset{\textcircled{c1}}{C(I^1)} \end{array}$$

23 $\quad \underset{\textcircled{b3}}{B(I^1, I^2)} = \qquad **2$

22 $\quad \underset{\textcircled{c2}}{C(I^1)} = \underset{\textcircled{b2}}{B(I^1 -1, I^2)}$

9. Add any extra variables necessitated by uses no longer appearing in their original statements:

$\quad X(I^1 + 1, I^2) = \underset{\textcircled{a2}}{A(I^1 + 1, I^2)}$

21 $\quad \underset{\textcircled{a1}}{A(I^1, I^2)} = \underset{\textcircled{b1}}{B(I^1, I^2)} + \underset{\textcircled{c1}}{C(I^1)}$

23 $\quad \underset{\textcircled{b3}}{B(I^1, I^2)} = X(I^1 + 1, I^2) **2$

22 $\quad \underset{\textcircled{c2}}{C(I^1)} = \underset{\textcircled{b2}}{B(I^1 -1, I^2)}$

This finally gives us the following rewriting of our original loop:

$\quad$ DO 24 $I^2 = 1, 5$

$\quad$ DO 24 SIM FOR ALL $I^1 \in \{x: 2 \leq x \leq 50\}$

$\quad X(I^1 + 1, I^2) = A(I^1 + 1, I^2)$

21 $\quad A(I^1, I^2) = B(I^1, I^2) + C(I^1)$

23 $\quad B(I^1, I^2) = X(I^1 + 1, I^2) **2$

22 $\quad C(I^1) = B(I^1 -1), I^2$

24 $\quad$ CONTINUE

V. Further Remarks

It is easy to deduce a general algorithm for the coordinate method from the preceeding example. The method can be extended to cover the case of an inconsistent ordering of the occurrences. In that case, the loop can be broken into a sequence of sub-loops. Every generation g for which the relation $g \ll g$ does not hold can be executed within a DO SIM loop. An algorithm for doing this is described in Chapter 4.

Observe that there are only $2^n-1$ choices of a coordinate projection $\pi$ for rewriting loop (1). It is easy to try them all, in decreasing order of the amount of parallel computation achieved, until one is found for which the rewriting is possible. Rule S1 should rapidly eliminate many choices.

It may happen that the rewriting cannot be done with any coordinate projection. In this case, a more general linear mapping $\pi$ must be sought, using the approach developed before for DO CONC loops. For example, no coordinate projection works for the relaxation loop (10).

V. Further Remarks

It is easy to deduce a general algorithm for the coordinate method from the preceeding example. The method can be extended to cover the case of an inconsistent ordering of the occurrences. In that case, the loop can be broken into a sequence of sub-loops. Every generation g for which the relation $g \ll g$ does not hold can be executed within a DO SIM loop. An algorithm for doing this is described in Chapter 4.

Observe that there are only $2^n$-1 choice[illegible] a coordinate projection $\pi$ for rewriting loop (1). It is easy to try them all, in decreasing order of the amount of parallel computation achieved, until one is found for which the rewriting is possible. Rule S1 should rapidly eliminate many choices.

It may happen that the rewriting cannot be done with any coordinate projection. In this case, a more general linear mapping $\pi$ must be sought, using the approach developed before for DO CONC loops. For example, no coordinate projection works for the relaxation loop (10).

# CHAPTER THREE

# PRACTICAL CONSIDERATIONS

## I. Restrictions on the Loop

Now consider the application of these methods to the problem of compiling a FORTRAN program for execution on a multiprocessor computer. We immediately observe that the restrictions which have been placed on the loop (1) would eliminate most real Fortran DO loops from consideration. For example, the DO limits $\ell^i$, $\mu^i$ are usually not all constants known at compile time. Fortunately, most of the restrictions were made to simplify the exposition, and are not essential. We will now describe the restrictions which are essential to the analysis.

First, some terms must be defined. By a "loop constant", we mean an expression whose value does not change while the loop is executed - i.e., any expression not involving generated variables or loop index variables. A quantity is "known at compile time" if it has a constant value which can be determined by the FORTRAN compiler.

The analysis can be applied to the following loop:

(13) DO $\alpha\ I^1 = \ell^1, \mu^1, d^1$

$\vdots$

DO $\alpha\ I^n = \ell^n, \mu^n, d^n$

$\boxed{\text{loop body}}$

$\alpha$ CONTINUE

assuming that it satisfies the following conditions:

1. Each $d^i$ is known at compile time.
2. The loop body contains no transfer of control to any statements outside it.
3. There is no I/O statement in the loop body.
4. For each subroutine or function call in the loop body, it is known which variable elements it can modify.
5. Each occurrence of a generated variable must be of the form VAR $(e^1, \ldots, e^m)$, with
$$e^i = a_1^i * I^1 + \ldots + a_n^i * I^n + c^i ,$$
where $c^i$ is a loop constant and each $a_j^i$ is known at compile time.

For the coordinate method, the following additional assumptions are required.

6. There is no transfer of control within the loop body.
7. For every generated variable VAR, each occurrence of VAR within the loop body must be of the form
$$\text{VAR}\ (a^1 * I^{j_1} + c^1, \ldots, a^m * I^{j_m} + c^m ),$$
where $c^i$ is a loop constant, $a^i = \pm 1$ or 0, and the $a^i$ and $j_i$ are the same for all occurrences of VAR.

By weakening the restrictions, many complicated details are added to the process of rewriting the loop. However, the analysis remains largely unchanged. Some of these details are described in Chapter 4.

A significant change is introduced by allowing occurrence mappings of the form given in 5. It necessitates a complicated restating of the Hyperplane and Plane Concurrency Theorems, as well as changing the method of choosing the mapping $\pi$. This will be discussed in a future paper.

Note that if the loop (13) satisfies these restrictions, then so does the inner DO $I^k$/.../ DO $I^n$ loop, for any k. (The index variables $I^j$ for $j < k$ are loop constants for the inner loop.)

## II. Meeting the Restrictions

Even if a given loop does not satisfy the above restrictions, it may be possible to rewrite it so that it does. We will give some useful techniques for doing this.

It is easy to fulfill the requirement that the DO statements be tightly nested. The method is illustrated by the following example. The loop

```
          DO 77 I¹ = 1, 10
16        A (I¹, 1) = 0
          DO 77 I² = 2, 20
               ⋮
```

can be rewritten as the following tightly nested loop:

```
          DO 77 I¹ = 1, 10
          DO 77 I² = 2, 20
16        IF (I² .EQ. 2) A(I¹, I²-1) = 0
               ⋮
```

This technique is referred to as quantifying statement 16. It may be possible to move the statement back outside the DO $I^2$ loop and unquantify it after the rewriting is performed.

Occurrence mappings can sometimes be rewritten by substituting for generated variables so that condition 5 is met. One trick is illustrated by the following example. Given

```
      K = N
      DO 6 I = 1, N
5     B(I) = A (K)
6     K = K - 1
```

We can rewrite it as

```
      DO 51 I = 1, N
5     B(I) = A (N + 1 - I)
51    CONTINUE
61    K = 1
```

This use of auxiliary variables to effect negative incrementing is fairly common in FORTRAN programs.

In condition 6, the real restriction is that there can be no possible loops inside the loop body. If this is the case, then transfer of control can easily be eliminated by quantifying assignment statements with logical IFs.

## III. Scalar Variables

Even though the loop satisfies all the restrictions, it is clear that these methods can give no parallel computation if there are generated scalar variables. Any such variable must be eliminated.

A common situation is for the variable to be just a temporary storage word within a single execution of the loop body. The variable X in the following loop is an example

```
      DO 3 I = 1, 10
      X = SQRT (A(I))
      B(I) = X
3     C(I) = EXP (X)
```

In this loop, each occurrence of X can be replaced by XX(I), where XX is a new variable.

In general, we want to replace each occurrence of the scalar by VAR($I^1, \ldots, I^n$), for a new variable VAR.* A simple analysis of the loop body's flow path determines if this is possible.

Another common situation is when the variable X appears in the loop body only in the statement

X = X + expression,

where the expression does not involve X. This statement just forms the sum of the expression for all points in the index set $\mathcal{J}$ . We can replace it by

*After the rewriting, to save space we can lower the dimension of V. by eliminating any subscript not containing a DO FOR ALL index variable.

the statement

$$VAR(I^1,\ldots,I^n) = \text{expression},$$

and add the following "statement" after the loop:

$$X = X + \sum_{(I^1,\ldots,I^n) \in \mathcal{L}} VAR(I^1,\ldots,I^n).$$

The sum can be executed in parallel with a special subroutine.

The same approach applies when the variable is used in a similar way to compute the maximum or minimum value of an expression for all points in $\mathcal{L}$ .

## IV. Conclusion

We have presented methods for obtaining parallel execution of a given DO loop. Many details and refinements were omitted for simplicity, but all the basic ideas have been included. Some of the methods are being implemented in the ILLIAC-IV FORTRAN compiler, as described in Chapter 4. Preliminary study indicates that they will yield parallel execution for a fairly large class of programs. This is true for other types of multiprocessor computers as well.

# CHAPTER FOUR

# THE ILLIAC IV TRANSLATOR

As a practical example of the use of our methods, we describe algorithms for translating a FORTRAN DO loop into an ILLIAC IV extended FORTRAN DO / DO FOR ALL loop. We thus adopt the syntax of this extended FORTRAN, as described in [4]. In particular, note that the "DO FOR ALL" statement is just our "DO SIM" statement.

## I. RESTRICTIONS

We make some restrictions on the given loop (13) in addition to restrictions 1-7 of Section 3-I.

8. No two variables appearing in the loop may be EQUIVALENCEd in any way.
9. There must be no subroutine calls in the loop body.
10. Functions called from inside the loop must not modify the value of any variable.
11. For each $j = 1, \dots, n$:

    (a) $\ell^j$ and $u^j$ must be of the form

    $$e + c_1 I^1 + \dots + c_{j-1} I^{j-1},$$

    where $e$ is a loop constant, and the $c_i$ are known at compile time.

    (b) For any $P$ in the index set, $\ell^j(P)$ and $u^j(P)$ must be positive.

    (c) If $| d^j | > 1$, then $\ell^j$ must be a loop constant.

Condition 11(b) means that for any $(P^1, \ldots, P^n)$ in the index set, the expressions $\ell^j$ and $u^j$ must be positive when $P^1, \ldots, P^{j-1}$ are substituted for $I^1, \ldots, I^{j-1}$.

## II. NOTATIONS

We now introduce some notations. Assume that we are given the loop (13). Let D denote the element $(d^1, \ldots, d^n) \in \mathbb{Z}^n$. It is called the increment vector of the loop. For any point $X = (x^1, \ldots, x^n) \in \mathbb{Z}^n$, we define $X * D \in \mathbb{Z}^n$ by

$$X * D = (x^1 d^1, \ldots, x^n d^n).$$

As before, we let $\mathcal{J}$ denote the index set of this loop.

We define a way of representing certain subsets of $\mathbb{Z}^n$ which was partially introduced before. We let "+" denote "any positive integer", and let "-" denote "any negative integer". We then have

$$\left(3, \begin{smallmatrix}+\\0\end{smallmatrix}, 6, -\right) = \{(3, x, 6, y) : x \geq 0,\ y < 0\}$$

$$\left(-2, \begin{smallmatrix}+\\0\\-\end{smallmatrix}, \begin{smallmatrix}0\\-\end{smallmatrix}\right) = \{(-2, x, y) : y \leq 0\}.$$

Note that $\begin{smallmatrix}+\\0\\-\end{smallmatrix}$ was previously denoted by "*".

We let $\phi$ denote the empty set.

Now let x and y be objects of any kind, and $d \in \mathbb{Z}$. Then we define

$$\Phi(x, y; d) = \begin{cases} x & \text{if } d \geq 0 \\ y & \text{if } d < 0 \end{cases}$$

$$\Lambda(x, y; d) = \begin{cases} y & \text{if } d \geq 0 \\ x & \text{if } d < 0 \end{cases}$$

Thus $\Phi(I^1, I^2 + N; -3) = I^2 + N$, etc. Note that $\Lambda(x,y; d) = \Phi(y, x; d)$.

The reason for this notation is shown by the following trivial result.

Theorem: Let $\underline{a}^i$, $a^i$, $\bar{a}^i \in \mathbb{Z}$, $i = 1, \ldots, n$, such that $\underline{a}^i \leq a^i \leq \bar{a}^i$. Let T be the linear function defined by

$$T[(I^1, \ldots, I^n)] = h_1 I^1 + \ldots + h_n I^n.$$

Then

$$\sum_i \Phi(\underline{a}^i, \bar{a}^i; h_i)\, h_i \leq T[(a^1, \ldots, a^n)] \leq \sum_i \Lambda(\underline{a}^i, \bar{a}^i; h_i)\, h_i ;$$

and these are the best possible bounds on $T[(a^1, \ldots, a^n)]$.

By "best possible bounds", we mean that given the $\underline{a}^i$, $\bar{a}^i$, we can choose the $a^i$ so that $T[(a^1, \ldots, a^n)]$ equals either of the bounds.

By the quantification of an occurrence, we mean the quantification of the statement containing the occurrence, as described in Section III-2. We say that an occurrence is quantified by $I^j$ if it is quantified by the expression $(I^j\ .EQ.\ \ell^j)$ or $(I^j\ .EQ.\ u^j)$.

## III. COMPUTING THE SETS $< f, g >$

Let f, g be occurrences of a variable, with occurrence mappings $T_f$, $T_g$. First, we must generalize our rules to cover the case of increments $d^j$ different from 1. An examination of the reasoning used in Section 1-V shows that this is accomplished by simply generalizing our definition of $< f, g >$ to the following:

$$< f, g > = \{X \in \mathbb{Z}^n : T_f(P) = T_g(P + X * D) \text{ for some } P \in \mathbb{Z}^n\} .$$

We can improve our results somewhat by considering smaller sets $< f, g >$. Namely, it is easy to see that everything we have done remains correct if we replace the set $< f, g >$ by any set containing

$$\{X \in \mathbb{Z}^n : T_f(P) = T_g(P + X * D) \text{ for some } P, P + X * D \in \mathcal{J} \text{ such that f is actually executed for the point P and g is executed for the point } P + X * D\}$$

as a subset.

We use this observation to obtain smaller sets $< f, g >$ in the following two ways:

(i) By using quantifications of the occurrences to replace "*" components of $< f, g >$ by more restricted components.

(ii) By using the bounds on the index set to replace $< f, g >$ by $\phi$ whenever possible.

As example of (i), consider the following statements in the body of loop (13), with $n = 2$ and $d^1 = d^2 = 1$:

$$\mathrm{IF}(I^1 \ .\mathrm{EQ}.\ \ell^1) \quad \underset{\textcircled{a1}}{A(I^2)} = \dots$$

$$\dots \underset{\textcircled{a2}}{A(I^2 + 1)} \dots$$

Then we can take $< a1, a2 >$ to be $(\genfrac{}{}{0pt}{}{+}{0}, -1)$ instead of $(*, -1)$. This is because a1 is only executed for points $(\ell^1, y) \in \mathcal{I}$, and a2 is executed for points $(w, z)$ with $w \geq \ell^1$.

As an example of (ii), consider the following loop:

```
      DO 22 I = 1, 10
22    B(I) = B(I + 10)
```

$$\underset{\textcircled{b1}}{B(I)} = \underset{\textcircled{b2}}{B(I + 10)}$$

We can let $< b1, b2 > = \phi$ rather than $(-10)$, because P and P $-10$ cannot both be in the index set for any P.

We now give an algorithm for computing $< f, g >$. The steps are discussed afterward.

1. Permute the subscript positions so that we have

$$T_f(I^1, \dots, I^n) = (a^1 I^{k_1} + f^{k_1}, \dots, a^r I^{k_r} + f^{k_r}, f_c^{r+1}, \dots, f_c^m)$$

$$T_g(I^1, \dots, I^n) = (a^1 I^{k_1} + g^{k_1}, \dots, a^r I^{k_r} + g^{k_r}, g_c^{r+1}, \dots, g_c^m)$$

with $k_1 < \dots < k_r$, and each $a^i = \pm 1$.*

2. If for some i, $f^i_c - g^i_c$ is known at compile time to be not equal to zero, then $< f, g > = \phi$. Otherwise,

3. For $j = 1, \dots, n$:

   (a) If $j = k_i$ for some i then:

      (i) If $f^j - g^j$ is known at compile time, then $x^j = (f^j - g^j) / (d^j \cdot a^i)$

      (ii) Otherwise, $x^j = \substack{+\\0\\-}$

   (b) If $j \neq k_i$ for all i, then define $x^j$ by the following table:

| | | g Quantified By: | | |
|---|---|---|---|---|
| | | $l^j$ .EQ. $\ell^j$ | $l^j$ .EQ. $u^j$ | Neither |
| f Quantified By: | $l^j$ .EQ. $\ell^j$ | 0 | + if $u^j \neq \ell^j$; $\substack{+\\0}$ if $u^j$ may equal $\ell^j$ | $\substack{+\\0}$ |
| | $l^j$ .EQ. $u^j$ | − if $u^j \neq \ell^j$; $\substack{0\\-}$ if $u^j$ may equal $\ell^j$ | 0 | $\substack{0\\-}$ |
| | Neither | $\substack{0\\-}$ | $\substack{+\\0}$ | $\substack{+\\0\\-}$ |

*A trivial modification to the algorithm handles the situation in which two of the $k_i$ are equal.

4. If for some j, $x^j$ is a number which is not an integer, then $< f, g > = \phi$. Otherwise,

5. If for some j, $x^j$ is an integer such that $|d^j \cdot x^j| > |u^j - \ell^j|$, then $< f, g > = \varphi$. Otherwise,

6. $< f, g > = (x^1, \ldots, x^n)$.

We now discuss the individual steps of the algorithm.

1. This step is possible because of Restriction 7 of Section III-1. It is only done to simplify the notation.

2. If $f_c^i \neq g_c^i$, then clearly $T_f(P) \neq T_g(Q)$ for all P, Q $\epsilon$ $Z^n$.

3. (a) (i) If $x^j$ is an integer, it is just the jth component of $< f,g >$. If it isn't an integer, then $< f, g > = \varphi$.

   (ii) In this case, we know that either $< f, g > = \phi$, or its jth components consists of an unknown integer. The best we can do is let the jth component of $< f, g >$ be $\overset{+}{\underset{-}{0}}$.

   (b) In this case, the jth component of $< f, g >$ is $\overset{+}{\underset{-}{0}}$. We use the quantification of f and g to refine this choice. In Subsection 4B, we will discuss how to decide whether or not $u^j$ may equal $\ell^j$.

4. This step tests for a non-integral $x^j$ produced in step 3(a) (i).

5. We will describe later a method for finding an upper bound for $u^j - \ell^j$.

## IV. THE COORDINATE METHOD

We now present algorithms for applying the coordinate method - described in Chapter II - to our translation problem. We assume that the loop appearing in the original FORTRAN program has been rewritten using the techniques described in Chapter 3 to make it satisfy our restrictions.

The given loop (13) will be rewritten as

```
(14)    DO α $I^{p_1} = \lambda^{p_1}, \mu^{p_1}, d^{p_1}$
                .
                .
                .
        DO α $I^{p_{n-k}} = \lambda^{p_{n-k}}, \mu^{p_{n-k}}, d^{p_{n-k}}$
        S = [ $(I^{j_1}, \ldots, I^{j_k}) / [\lambda^{j_1}, \lambda^{j_1} + d^{j_1} \ldots \mu^{j_1}]$ .CROSS. ...
X               .CROSS. $[\lambda^{j_k}, \lambda^{j_k} + d^{j_k} \ldots \mu^{j_k}]$ : condition 1 .AND.
X               ... .AND. condition m]
        DO α FOR ALL $(I^{j_1}, \ldots, I^{j_k}) / S$ ,
```

where $p_1 < \ldots < p_{n-k}$ and $j_1 < \ldots < j_k$. (For notational convenience, we have replaced the k of Chapter 3 by n-k.) The conditions in the set expression are boolean expressions. "S" denotes some unique identifier.

The mapping $\pi : \mathbb{Z}^n \to \mathbb{Z}^{n-k}$ of Chapter II is thus defined by

$$\pi [ (I^1, \ldots, I^n) ] = (I^{p_1}, \ldots, I^{p_{n-k}}) .$$

The following data declarations must also appear

$$\text{SET } S\,(\Lambda(\lambda^{j_1}, \mu^{j_1}; d^{j_1}), \ldots, \Lambda(\lambda^{j_k}, \mu^{j_k}; d^{j_k}))$$

$$\text{ALLOCATE } S\,((1, 2, \ldots, k))\ .$$

(Since the $d^j$ are known at compile time, the $\Lambda$s are evaluated by the translator and do not actually appear in the translator's output.)

A. Rewriting the DO Statements

We first give an algorithm for finding the limits $\lambda^i$, $u^i$ and the conditions defining S in (14). Note that the $\lambda^{j_i}$, $u^{j_i}$ must be integer constants. We assume that for the original loop (13), we have

$$(15)\qquad \ell^j = \ell_0^j + \ell_1^j I^1 + \ldots + \ell_{j-1}^j I^{j-1}$$

$$u^j = u_0^j + u_1^j I^1 + \ldots + u_{j-1}^j I^{j-1}\ ,$$

as described in Restriction 11 of Section I. The steps in the algorithm are explained afterwards.

The Algorithm

1. For j = 1, ..., n; compute the following loop constant expressions:

$$\underline{\ell}^j = \ell_0^j + \sum_{i=1}^{j-1} \ell_i^j\ \Phi(\underline{\ell}^i, \overline{u}^i; d^i \cdot \ell_i^j)$$

$$\overline{u}^i = u_0^j + \sum_{i=1}^{j-1} u_i^j\ \Lambda(\underline{\ell}^i, \overline{u}^i; d^i \cdot u_i^j)$$

$$\underline{(u - \ell)}^j = \Phi(1, -1; d^j) \; [u_0^j - \ell_0^j + \sum_i (u_i^j - \ell_i^j) \; \Phi(\underline{\ell}^i, \overline{u}^i; d^i (u_i^j - \ell_i^j))]$$

$$(\overline{u - \ell})^j = \Phi(1, -1; d^j) \; [u_0^j - \ell_0^j + \sum_i (u_i^j - \ell_i^j) \; \Lambda(\underline{\ell}^i, \overline{u}^i; d^j (u_i^j - \ell_i^j))]$$

2. For $j = 1, \ldots, n$;

   $D^j$ = minimum { $a^s$: There is an occurrence in the loop, not quantified by $I^j$, such that $I^j$ appears in the sth subscript position of the occurrence mapping, for a variable whose dimensions are $(a^1, \ldots, a^n)$ . }

   $D^j$ is undefined if this set is empty.

3. If $D^{j_i}$ is undefined for any $i = 1, \ldots, k$; then the rewriting is impossible.

4. For $i = 1, \ldots, k$:

   (a) $\Phi(\lambda^{j_i}, u^{j_i}; d^{j_i}) = 1$

$$\Lambda(\lambda^{j_i}, u^{j_i}; d^{j_i}) = \begin{cases} 1 + (\overline{u - \ell})^j, & \text{if this is known at compile time} \\ D^{j_i} & \text{otherwise} \end{cases}$$

(b) Throughout the original loop, and in (15), replace $I^{j_i}$ by $I^{j_i} + \underline{\ell}^{j_i} - \lambda^{j_i}$, $\ell_0^{j_i}$ by $\ell_0^{j_i} - \underline{\ell}^{j_i} + \lambda^{j_i}$ and $u_0^{j_i}$ by $u_0^{j_i} - \underline{\ell}^{j_i} + \lambda^{j_i}$. I.e., recompute the $\ell_0^j$ and $u_0^j$, but <u>not</u> the quantities computed in step 1.

(c) If $\ell^{j_i}$ is not known at compile time, add the condition

$(\ell^{j_i}\ \Phi(\text{.LE.}\ ,\ \text{.GE.}\ ;\ d^{j_i})\ I^{j_i})$ .

(d) If $u^{j_i}$ is not known at compile time, add the condition

$(u^{j_i}\ \Phi(\text{.GE.}\ ,\ \text{.LE.}\ ;\ d^{j_i})\ I^{j_i})$. Otherwise, set $u^{j_i} = u^{j_i}$

(Note: This is the new value of $u^{j_i}$ computed in (b).)

5. For $i = 1, \ldots, n - k$:

(a) Set $\lambda^{p_i}$ equal to $\ell^{p_i}$, with

$\Phi(\lambda^{j_r}, u^{j_r}; d^{j_r} \cdot \ell_{j_r}^{p_i})$

substituted for each $I^{j_r}$ , $r = 1, \ldots, k$.

(b) Set $u^{p_i}$ equal to $u^{p_i}$, with

$\Lambda(\lambda^{j_r}, u^{j_r}, d^{j_r} \cdot \ell_{j_r}^{p_i})$

substituted for each $I^{j_r}$.

(c) For $r = 1, \ldots, k$:

(i) If $\ell_{j_r}^{p_i} \neq 0$, $j_r < p_i$; then add the condition

$(\ell^{p_i}\ \Phi(\text{.LE.}\ ,\ \text{.GE.}\ ;\ d^{p_i})\ I^{p_i})$.

(ii) If $u_{j_r}^{p_i} \neq 0$, $j_r < p_i$ ; then add the condition

$$(u^{p_i} \Phi(.GE., .LE.; d^{p_i}) I^{p_i}) .$$

6. Let i be the smallest integer such that the conditions defining S do not depend upon the values of $I^{p_i}, I^{p_{i+1}}, \ldots, I^{p_{n-k}}$. If i exists, then place the assignment statement for S just before the DO $\alpha$ $I^{p_i}$ statement. Otherwise, place it after the DO $\alpha$ $I^{p_{n-k}}$, as it appears in (14).

Explanation of the Algorithm

1. If $d^j > 0$, then by the theorem of Section II, $\underline{\ell}^j \leq I^j(P) \leq \overline{u}^j$ for all $P \in \mathcal{D}$. If $d^j < 0$, then the inequalities are reversed.

   Similarly, $(\underline{u - \ell})^j$ and $(\overline{u - \ell})^j$ are chosen so that $(\underline{u - \ell})^j \leq | u^j(P) - \ell^j(P) | \leq (\overline{u - \ell})^j$ for all $P \in \mathcal{D}$. Although the $(\underline{u - \ell})^j$ are not used in this algorithm, this is an appropriate place to calculate them. They will be used later.

2. The number $D^j$ provides an upper bound for $1 + | u^j(P) - \ell^j(P) |$ ; $P \in \mathcal{D}$. To see this, suppose the occurrence $A(I^2, I^3 + K)$ appears in the loop, and A is dimensioned by

   DIMENSION A(10, 33).

If references occur to both A(i,j) and A(i', j'), then $| i - i' | \leq 10 - 1 = 9$, $| j - j' | \leq 32$. Hence, $| u^3(P) + K - (\ell^3(P) + K) | = | u^3(P) - \ell^3(P) | \leq 33 - 1$.

3. $D^{j_i}$ is undefined if $I^{j_i}$ does not appear in any unquantified variable occurrences. If this is the case, then we have made a bad choice of DO FOR ALL variables (i.e., of the mapping $\pi$).

4. (a) Suppose $d^{j_i} > 0$. Then $u^{j_i} - \lambda^{j_i}$ is an upper bound for $u^{j_i}(P) - \ell^{j_i}(P)$, $P \in \mathcal{O}$. If $d^{j_i} < 0$, reverse the signs in the preceding statement. Note that $\lambda^{j_i}$ and $u^{j_i}$ are actual numbers, known at compile time.

   (b) We rewrite the loop so that the values assumed by $I^{j_i}$ are as small as we can make them and still be sure that they are positive.

   Note that if $\ell^{j_i}$ was originally a loop constant, then after the substitution it equals $\lambda^{j_i}$, a known number. This is necessary if $| d^{j_i} | > 1$, otherwise our set construction would not work right. I.e., the set $[ \lambda^{j_i}, \lambda^{j_i} + d^{j_i} \ldots {}^{j_i}]$ would not necessarily contain integers in the correct congruence class mod $d^{j_i}$. This is the reason for restriction 11 (c) of Section I.

   (c), (d) If $\lambda^{j_i}$ or $u^{j_i}$ is not the actual value of $\ell^{j_i}$ or $u^{j_i}$ respectively, then the appropriate tests must be inserted.

5. If the limits on $I^{p_i}$ depend upon one or more $I^{j_r}$, then the bounds $\lambda^{j_r}$, $u^{j_r}$ on $I^{j_r}$ must be used to determine the new limits on $I^{p_i}$, and the original limit test must appear in the definition of S. Note that the expressions $\ell^{p_i}$, $u^{p_i}$ are the ones recomputed in Step 4(b).

6. This just removes loop-invariant code from the innermost DO loops.

B. <u>The Bounds on $| u^j - \ell^j |$</u>

The algorithm of Subsection A finds the bounds on $| u^j - \ell^j |$ needed for the algorithm of Section 3 which constructs the sets < f, g >.

For Step 3 of the algorithm of Section III, we can say that $u^j \neq \ell^j$ if and only if $(\underline{u - \ell})^j$ is known at compile time, and is not equal to zero.

For Step 5 of that algorithm, we may replace $| u^j - \ell^j |$ by $(\overline{u - \ell})^j$ if it is known at compile time, or by $D^j$ if it is defined. If neither replacement is possible, then the test may not be applied for that value of j.

Note that the values of $(\underline{u - \ell})^j$ and $(\overline{u - \ell})^j$ are not changed by the substitution for the $I^{j_i}$ performed in Step 4(b) of the algorithm of Subsection A.

## C. The Coordinate Algorithm

We now describe a complete algorithm for rewriting the given loop (13) as a DO / DO FOR ALL loop of the form (14). Actually, we will "un-tightly nest" the loop if possible by moving quantified statements outside of inner loops. (See Section 3-II.) A discussion of the individual steps follows the description of the algorithm.

The Algorithm

1. Find the sets < f, g > required by rules S1 and S2, using the algorithm of Section 3.
2. Choose the DO FOR ALL variables $I^{j_1}, \ldots, I^{j_k}$.
3. Apply the algorithm of Subsection A to rewrite the DO statements. If the rewriting is found to be impossible, stop.
4. Apply rules S1-S3 of Chapter 2 to obtain the ordering relations <<. If the rewriting is found to be impossible, stop.
5. Replace the ordering << by its transitive closure. I.e., add the relations f << g which are implied by transitivity from the original ordering. Thus, if f << h << g, then add the relation f << g.
6. If for some generation g, the relation g << g holds, then go to the algorithm described in Subsection E.
7. Complete the ordering of the generations to a total ordering by the following procedures. Let f and g be generations which are unordered with respect to one another (neither f << g nor g << f

holds). Then:

(a) With the notation of (14): if, for some i, f is quantified by $(I^{p_i} .EQ. \ell^{p_i})$ and g is not, then f << g.

(b) If, for some i, f is quantified by $(I^{p_i} .EQ. u^{p_i})$ and g is not, then g << f.

(c) If the statements containing f and g are both of the form IF (expression), with the same "expression", then make f and g adjacent in the complete ordering. (I.e., if we make f << g, then f << h << g must imply h = f or h = g.)

After applying (a), (b), and (c) for all pairs of unordered generations, complete the ordering by using the order in which the generations appear in the original loop.

Note: No new ordering relations involving uses are added.

8. Denote the generations by $g_1, \ldots, g_m$, with $g_1 << \ldots << g_m$. Reorder the statements of the loop body so that the ith statement is the one in which $g_i$ appears, for $i = 1, \ldots, m - 1$. (Note that by our restrictions, every loop body statement contains a generation.)

For notational convenience, introduce an imaginary generation $g_0$ with $g_0 << f$ for every occurrence f in the loop, and a corresponding Statement 0.

9. For each variable use f, let

$\text{first}\ (f) = \text{maximum}\ \{ i : g_i << f \}$

$\text{last}\ (f) = \text{minimum}\ \{ i : f << g_i \}$

Note that if f appears in Statement j, then

$0 \leq \text{first}\ (f) < \text{last}\ (f) \leq j.$

10. For j = 1, ..., m:

(a) For each use f appearing in Statement j:

(i) If last (f) = j, leave f unchanged.

(ii) If last (f) < j, then let $\overline{f}$ be a variable occurrence with the same occurrence mapping as f, but with a different, previously unused variable name. Replace f by $\overline{f}$ in statement j. Form an assignment statement $\overline{f} = f$, with the same quantifications ($I^{P_i}$ .EQ. $\ell^{P_i}$ or $u^{P_i}$) as statement j. For example, if statement j is

IF (($I^{P_2}$ .EQ. $\ell^{P_2}$) .AND. ($I^{j_1}$ .EQ. $\ell^{j_1}$)) ....,

form the statement

IF ($I^{P_2}$ .EQ. $\ell^{P_2}$) $\overline{f} = f$.

Insert this new statement anywhere between statements first (f) and last (f).

(b) If Step (a) (ii) was applied to two distinct but identical uses $f_1$, $f_2$ (i.e., uses of the same variable having the same occurrence mappings), then if possible, replace $\overline{f}_1$ and $\overline{f}_2$ by a single occurrence $\overline{f}$, with a single generation.

11. (a) If the first q statements of the loop body (including those added in step (10)) are all quantified by $(I^{p_j}$ .EQ. $\ell^{p_j})$ for $j = i, i + 1, \ldots, n - k$, but the $(q + 1)^{\underline{st}}$ statement is not then:

    (i) Move these q statements in front of the DO $I^{p_i}$ statement.

    (ii) For $j = 1, \ldots, n - k$: Delete the $(I^{p_j}$ .EQ. $\ell^{p_j})$ quantifiers, and replace all instances of $I^{p_j}$ by $\ell^{p_j}$ in these statements.

    (iii) Insert a

    DO β FOR ALL $(I^{j_1}, \ldots, I^{j_k})$ / S·

    statement in front of these q statements and a

    β CONTINUE

    statement after them.

    (iv) If the assignment statement for S lies inside the DO $I^{p_i}$ loop: then copy the assignment statement for S, except with $I^{p_j}$ replaced by $\ell^{p_j}$ for $j = i, i + 1, \ldots, n - k$.

    (v) Repeat (a).

(b) If the last q statements of the loop body are all quantified by ($I^{p_j}$ .EQ. $\ell^{p_j}$) for j = i, ..., n - k, but the (q + 1)st statement is not, then perform the analogue of steps (a) (i) - (v) to place these statements after the DO $I^{p_i}$ loop.

12. For each variable introduced in Step 10, and each variable introduced by the procedures of Section 3-III:

(a) For each occurrence of the variable: Delete all subscript positions not referencing any $I^{j_i}$. E.g., replace VAR ($I^{j_1}$ + 2, $I^{p_2}$, $I^{j_3}$, 4) by VAR ($I^{j_1}$ + 2, $I^{j_3}$).

(b) Create the appropriate data type and dimension statements. The actual dimensions are obtained by:

(i) For the set S introduced in Step 3, and the variables introduced as described in Section 3-III, the ith dimension is $\Lambda(\lambda^{j_i}, u^{j_i}; d^{j_i})$, where $\lambda^{j_i}$, $u^{j_i}$ were found in Step 3.

(ii) For the variables introduced in Step 10, the dimensions are obtained in the obvious way from the dimensions of the variables which they replace.

(c) Create the appropriate ALLOCATE command needed to make the DO FOR ALL valid.

(d) Place the variable in an OVERLAP statement so as to allow it to occupy the same storage area as other variables similarly introduced in other loops.

13. For each original loop variable VAR which does not have the proper storage allocation for the DO FOR ALL, introduce a new variable $\overline{\text{VAR}}$ of the same dimensions, but appropriate allocation. Add the statement

$$\overline{\text{VAR}} = \text{VAR}$$

in the front of the DO FOR ALL loop. If VAR is a generated variable, add

$$\text{VAR} = \overline{\text{VAR}}$$

after the DO FOR ALL loop. Replace all instances of VAR in the loop by $\overline{\text{VAR}}$.

<u>Discussion of the Algorithm</u>

2. Some heuristic will be needed for choosing the DO FOR ALL variables, probably involving the sets < f, g > computed in Step 1.

5. See [5] for an algorithm to do this.

6. If g << g holds for some g, then the generation g cannot be executed inside the DO FOR ALL loop. However, as we will see, the DO FOR ALL may be applied to any other generation h for which the relation h << h does <u>not</u> appear.

7. We need a total ordering of the statements so that we can rewrite the loop body. I.e., we need to know the order in

which to write down the statements. This is done by totally ordering the generations. (Since there are no control statements, there is exactly one generation per statement.) If Step 5 did not provide a total ordering, we must add ordering relations until we get one.

Steps (a) and (b) try to maximize the chances of being able to use Step 11. Step (c) tries to make the handling of mode sets as easy as possible for the compiler.

If these steps do not determine a total ordering, using the order of appearance in the loop seems as good a method as any to complete the ordering.

This entire step is not very precisely formulated, but should indicate how an algorithm for executing it can be obtained.

9. The use f must be executed between generation numbers first (f) and last (f). If first (f) $\geq$ last (f), then there would be a generation g with g << g.

10. (a) (ii) In this case the use f must be executed before the value it produces is needed. I.e., the load must be executed before statement j is executed. Hence, a temporary storage location is needed.

(b) The conditions for combining two temporary storage locations are clear, although when to combine IF clauses, etc., requires a detailed algorithm. Note that a necessary condition for being able to combine the assignment statements is that there exists an $r$ such that

$$\text{first }(f_i) < r \leq \text{last }(f_i)$$

for $i = 1, 2$.

Even if this is not possible, the same variable can sometimes be used, with two generations, thus saving storage space.

A method of minimizing storage space which we have not considered is illustrated by the following example. Suppose we have

$$A(I^1) = B(I^1) + C(I^1)$$

and (a) (ii) indicates that we need temporary variables for these uses of B and C. It may be possible to rewrite this as

$$\overline{BC}\ (I^1) = B(I^1) + C(I^1)$$

$$\vdots$$

$$A(I^1) = \overline{BC}\ (I^1)\ .$$

Developing an algorithm to do this is rather involved, and has not been done.

11. This is a fairly obvious procedure for moving statements outside of inner loops. It actually reverses the procedure for moving them into inner loops described in Section 3-II. As the example given in Subsection D shows, what gets moved

inside a loop cannot always be moved back outside again.

If (a) and (b) both move statements outside the same DO $I^{p_i}$ loop, then there is no need to compute the same set S twice. However, a new set variable must be used in place of S in steps (iii) and (iv).

12. (a) This is an obvious space saving technique.

(b) Our algorithm has neglected the possibility of overlapping the variables introduced in Steps 10 and 13 with each other. A trivial analysis shows when two such variables may be overlapped. Finding the optimal overlapping among a collection of such variables requires a clever algorithm, which has not been developed.

13. This is an obvious solution to the reallocation problem. In some cases it would be better to put the reallocation inside one or more of the outer DO $I^{p_i}$ loops, and set $\overline{\text{VAR}}$ equal to the appropriate "coordinate slice" of VAR. However, we have not devised an algorithm to do this.

## D. An Example

We illustrate the preceding algorithm by applying it to the following unlikely loop:

DO 17 $I^1 = 2, N$

DO 17 $I^2 = I^1, 1, -1$

13 IF $(I^2\ .EQ.\ I^1)\ \underset{\text{B1}}{B(I^1)} = \underset{\text{A1}}{A(I^2, I^1 - 1)} + SIN\ (3.14 * I^1 / 180)$

14 $\underset{\text{A2}}{A(I^2, I^1)} = \underset{\text{C1}}{C(I^1, I^2 + K)} + .5$

15 $\underset{\text{C2}}{C(I^1, I^2)} = \underset{\text{A3}}{A(I^2, I^1 + 1)} * \underset{\text{B2}}{B(I^1)}$

16 IF $(I^2\ .EQ.\ 1)\ \underset{\text{E1}}{E(I^1)} = \underset{\text{C3}}{C(I^1, I^2)}$

17 CONTINUE.

with the following dimension information:

DIMENSION A(4, 95), B(97), C(99, 8), E(101)

Note that statements 13 and 16 were originally outside the inner DO $I^2$ loop, but were moved inside it by the method of Section 3-II.

We now apply the algorithm as follows :

1. The algorithm of Section III yields the following:

$< A1, A2 > \;=\; (-1, 0)$

$< A2, A2 > \;=\; (0, 0)$

$< A2, A3 > \;=\; (-1, 0)$

$< B1, B1 > \;=\; (0, 0)$

$< B1, B2 > \;=\; (0, \overset{+}{0})$ [using step 3(b)]

$< C1, C2 > \;=\; (0, \overset{+}{\underset{-}{0}})$ [using step 3(a) (ii) ]

$< C2, C3 > \;=\; (0, 0)$

$< E1, E1 > \;=\; (0, 0)$ .

2. We choose to rewrite the loop with a DO FOR ALL $I^1$, so we let $j_1 = 1$ and $p_1 = 2$. Then

$$\pi [ (I^1, I^2) ] = I^2 .$$

3. We apply the algorithm of Subsection A as follows:

   1. $\underline{\ell}^1 = 2$, $\bar{u}^1 = N$, $(\underline{u - \ell})^1 = N - 2$, $(\overline{u - \ell})^1 = N - 2$

      $\underline{\ell}^2 = N$, $\bar{u}^2 = 1$, $(\underline{u - \ell})^2 = 1$ , $(\overline{u - \ell})^2 = N - 1$

   2. $D^1 = \text{minimum } \{ 95, 97, 99, 101 \} = 95$

      $D^2 = \text{minimum } \{ 4, 8 \} = 4$

   3. $D^1$ is defined, so no problem.

   4. (a) $\lambda^1 = 1$

      $\mu^1 = 95$

      (b) Substitute $I^1 + 1$ for $I^1$ throughout the loop, and change $\ell^i$, $u^i$ as follows:

$$\ell^1 = 1,\ u^1 = N - 1$$

$$\ell^2 = 1 + I^1,\ u^2 = 1$$

(c) $\ell^1$ known at compile time, no condition

(d) Add the condition ((N - 1) .GE. $I^1$) .

5. (a) $\lambda^2 = 96$ (= 1 + $u^1$)

(b) $u^2 = 1$

(c) (i) Add the condition ((1 + $I^1$) .GE. $I^2$)

(ii) No condition

6. The conditions defining S involve $I^2$, so the assignment statement for S follows the DO 17 $I^2$ statement.

Putting this all together, we get the loop control statement

DO 17 $I^2$ = 96, 1, -1

S = [ $I^1$ / [1, 2 ... 95] : ((N - 1) .GE. $I^1$) .AND.
((I$^1$ + 1) .GE. $I^2$) ]

DO 17 FOR ALL $I^1$ / S

and the new loop body:

13 IF ($I^2$ .EQ. $I^1$ + 1) $\underset{\text{B1}}{\text{B}(I^1 + 1)}$ = $\underset{\text{A1}}{\text{A}(I^2,\ I^1)}$ + SIN (3.14 * $I^1$ / 180)

14 $\underset{\text{A2}}{\text{A}(I^2,\ I^1 + 1)}$ = $\underset{\text{C1}}{\text{C}(I^1 + 1,\ I^2 + \text{K})}$ + .5

15 $\underset{\text{C2}}{\text{C}(I^1 + 1,\ I^2)}$ = $\underset{\text{A3}}{\text{A}(I^2,\ I^1 + 2)}$ * $\underset{\text{B2}}{\text{B}(I^1 + 1)}$

16 IF ($I^2$ .EQ. 1) $\underset{\text{E1}}{\text{E}(I^1 + 1)}$ = $\underset{\text{C3}}{\text{C}(I^1 + 1,\ I^2)}$

17 CONTINUE .

Note that none of the sets < f, g > computed in Step 1 are changed by the substitution for $I^1$.

We must also include the declarations

SET S(95)

DIMENSION $\overline{A}$(95) ,

and add S, $\overline{A}$ to an OVERLAP statement.

4. The rules give the following relations:

S1: A2 << A1

A3 << A2

(and the rewriting is not impossible)

S2: B1 << B2

C1 << C2

C2 << C3

S3: A1 << B1

C1 << A2

A3 << C2

B2 << C2

C3 << E1

5. This gives the following additional ordering relations:

| | |
|---|---|
| A2 << B1 | [from A2 << A1, A1 << B1] |
| C2 << E1 | [C2 << C3 << E1] |
| B1 << E1 | [B1 << B2 << C3 << E1] |
| A2 << E1 | [A2 << B1 << E1] |
| B1 << C2 | [B1 << B2 << C2] |

6. g << g does not hold for any generation g.

7. The ordering of the generations obtained by 4 and 5 is already totally ordered. It is: A2 << B1 << C2 << E1.

8. $g_1$ = A2, $g_2$ = B1, $g_3$ = C2, $g_4$ = E1

   This assigns the numbers 1-4 to the FORTRAN statements as follows:

   14 ← 1, 13 ← 2, 15 ← 3, 16 ← 4.

9. first (A1) = 1, last (A1) = 2

   first (C1) = 0, last (C1) = 1

   first (A3) = 0, last (A3) = 1

   first (B2) = 2, last (B2) = 3

   first (C3) = 3, last (C3) = 4

10. (a) The only use for which (ii) holds is A3. We must change statement 15 which contains A3 to (recall that we are using the loop as rewritten in Step 3):

    15 $C(I^1 + 1, I^2) = \overline{A}(I^2, I^1 + 2) * B(I^1 + 1)$

    and add the following statement before the first statement of the loop body

    $\overline{A}(I^2, I^1 + 2) = A(I^2, I^1 + 2)$

    (b) Does not apply.

11. (a) We may remove statement 16 from the outer DO $I^2$ loop, with its own DO FOR ALL, as follows:

```
         S = (I^1 / [1, 2 ... 95] : ((N - 1) .GE. I^1) .AND.
             ((I^1 + 1) .GE. 1))
         DO 116 FOR ALL I^1 / S
16       E(I^1 + 1) = C(I^1 + 1, 1)
116      CONTINUE
```

12. (a) We replace the two occurrences $\overline{A}(I^2, I^1 + 2)$ introduced in Step 10 by $\overline{A}(I^1 + 2)$.

(b) We need the following statements

```
SET S(95)
DIMENSION Ā(95)
```

(c) Since we have only added 1-dimensional variables, no ALLOCATE statements need be added.

(d) We will have an OVERLAP statement of the form

```
OVERLAP ... (S, Ā) ...
```

13. Assuming default allocations, this step is vacuous.

Putting this altogether now, we finally get the following loop:

```
      DO 17 I^2 = 96, 1, - 1
      S = [I^1 /]1, 2 ... 95] : ((N - 1) .GE. I^1) .AND.
          ((I^1 + 1) .GE. I^2) ]
      DO 17 FOR ALL I^1 / S
      Ā (I^1 +2) = A(I^2, I^1 + 2)
14    A(I^2, I^1 + 1) = C(I^1 + 1, I^2 + K) + .5
```

13 $\quad$ IF($I^2$ .EQ. $I^1 + 1$) B($I^1 + 1$) = A($I^2$, $I^1$) + SIN (3.14 * $I^1$ / 180)

15 $\quad$ C($I^1 + 1$, $I^2$) = $\bar{A}$($I^1 + 2$) * B($I^1 + 1$)

17 $\quad$ CONTINUE

S = [$I^1$ / [1, 2 ... 95] : ((N - 1) .GE. $I^1$) .AND.
    (($I^1$ + 1) .GE. 1)]

DO 116 FOR ALL $I^1$ / S

16 $\quad$ E($I^1 + 1$) = C($I^1 + 1$)

116 $\quad$ CONTINUE

## E. The Algorithm for Inconsistent Orderings

We now give an algorithm to handle the situation in which Step 4 or 5 of the coordinate algorithm of Subsection C produces an inconsistent ordering - i.e., one containing a relation of the form $g \ll g$. The algorithm is discussed afterwards.

### The Algorithm

1. Define an equivalence relation $\backsim$ on the set of generations by $f \backsim g$ if and only if (i) $f = g$, or (ii) $f \ll g \ll f$.

   Let $[f]$ denote the equivalence class of f. I.e.,

   $[f] = \{ g : g \backsim f \}$ .

   Let $\mathcal{G}$ denote the set of equivalence classes.

   Define an order relation on $\mathcal{G}$ by $[f] \ll [g]$ if and only if $[f] \neq [g]$ and $f \ll g$. (Note that this is independent of the choice of $f \in [f]$, $g \in [g]$.) This ordering is transitively, closed (because the ordering of generations is) and is consistent. I.e., $[g] \not\ll [g]$ for all generations g.

Call the class [f] good if $f \not\ll f$, and bad if $f < f$. (Again, this is independent of the choide of $f \in [f]$.)

2. Write $\mathcal{G}$ as the disjoint union of subsets $\mathcal{G}_1, \ldots, \mathcal{G}_m$; where m is as small as possible, and the $\mathcal{G}_i$ satisfy:

    (i) The elements of $\mathcal{G}_i$ are either all good or all bad.

    (ii) If $[f] \in \mathcal{G}_i$, $[g] \in \mathcal{G}_j$ and $i < j$, then $[f] \not\gg [g]$.

    Let $G_i = \{ g : [g] \in \mathcal{G}_i \}$. Call $G_i$ good if for each $g \in G_i$ [g] is good, and call it bad if [g] is bad for each $[g] \in G$. Then (i) states that $G_i$ is either good or bad. The minimality of m implies that if $G_i$ is good, then $G_{i-1}$ and $G_{i+1}$ are bad.

3. Totally order the generations as follows:

    (a) If $G_i$ is good, use Step 7 of the Coordinate algorithm of Subsection C, minus parts (a) and (b), to order the elements of $G_i$.

    (b) If $G_i$ is bad, order the elements of $G_i$ by the order in which they appear in the original loop.

    (c) If $f \in G_i$, $g \in G_j$ and $i < j$, then $f << g$. (Note that property (ii) of the $\mathcal{G}_i$ in Step 2 assures that this gives a consistent ordering.)

4. For each $G_i$, let $\overline{G}_i$ be the set of all occurrences f such that f appears in the same statement as g for some $g \in G_i$. If

$G_i$ is a bad set and $g \in G_i$, let $\bar{\bar{G}}_i = \{ f : f$ a use and $f \in \bar{G}_i$ or $f < g < f \}$

5. For each bad set $G_i$: Delete all relations of the form $f \ll g$, $f, g \in \bar{\bar{G}}_i$, which were found by rule S1 but not by rules S2 or S3 (applied in Step 6 of the Coordinate Algorithm).

All the remaining ordering relations $f \ll g$, including those defined in Step 3, form a consistent ordering of the occurrences.

6. For each bad set $G_i$, and each use $f \in \bar{\bar{G}}_i - \bar{G}_i$, introduce a new k-dimensional variable $\bar{A}$ and add the statement

$$\bar{A}\,(I^{j_1}, \ldots, I^{j_k}) = f$$

Let $\bar{f}$ denote this generation of $\bar{A}$. Add $\bar{f}$ to $G_i$, add the ordering $f \ll \bar{f}$, and totally order $G_i$ so that the new ordering (of all occurrences) is consistent. Finally, replace the original use f by $\bar{A}\,(I^{j_1}, \ldots, I^{j_k})$.

7. Perform Steps 8, 9, 10, and 12 of the Coordinate Algorithm. Add the generations inserted by Step 10 to the appropriate sets $G_i$ (maintaining 3(c)). Whenever possible, these new statements should be inserted so that their generations are included in good sets.

8. Let $\mathcal{S}_i$ be the statements containing the generations in $G_i$, ordered by the ordering between the generations.

9. Rewrite the loop as an outer DO $I^{p_1}$, ..., DO $I^{p_k}$ loop containing the assignment statement for S (as constructed in Step 3 of the Coordinate Algorithm of Subsection C), and a sequence of m inner loops as follows.

For $i = 1, \ldots, m$:

(a) If $G_i$ is a good set, then insert a

```
DO FOR ALL ($I^{j_1}$, ..., $I^{j_k}$) / S
```

loop whose body consists of the statements $\mathcal{S}_i$.

(b) If $G_i$ is a bad set, then insert the loop

```
DO β $I^{j_1} = \ell^{j_1}, u^{j_1}, d^{j_1}$
        .
        .
        .
DO β $I^{j_k} = \ell^{j_k}, u^{j_k}, d^{j_k}$
IF (.NOT. (condition1 .AND. ... .AND. condition p)) GO TO β
    $\mathcal{S}_i$
β CONTINUE,
```

where the conditions are those chosen in Step 5(c) of the algorithm of Subsection A (which was executed in Step 3 of the Coordinate Algorithm).

10. Perform Step 13 of the Coordinate Algorithm.

Remarks on the Algorithm

The fact that the algorithm is correct follows from the following observation:

The rewritten loop will be computationally equivalent to the original one if for every relation f << g produced by the rules S1-S3, one of the following holds:

(i) f appears in $\mathcal{S}_i$, g appears in $\mathcal{S}_j$, with i < j.

(ii) f and g appear in the same DO FOR ALL loop, and precedes g.

(iii) f and g appear in the same DO loop, in the same order in which they appeared in the original loop.

(iv) f and g appear in the same DO loop, and f precedes g.

The only ones of the above conditions which are not obviously sufficient are (iii) and (iv). For (iv), observe that the order of execution of f and g only matters if $\vec{0} \in \langle f, g \rangle$ and originally f << g or g << f. However, if $\vec{0} \in \langle f, g \rangle$ and originally g <<f, then rule S2 produces g << f, so the above conditions cannot hold for <u>all</u> f and g. Hence, if $\vec{0} \in \langle f, g \rangle$, then f << g and (iv) implies (iii).

For (iii), let J be the mapping we defined in Section I-3, which for the loop (14) is given by

$$J[(I^1, \ldots, I^n)] = (I^{P_1}, \ldots, I^{P_{n-k}}, I^{j_1}, \ldots, I^{j_k}) .$$

Then the order of execution of references by f and g to the same variable element are reversed only if for some $A \in \langle f, g \rangle$, and $A > \vec{0}$ and $J(A) < \vec{0}$, or $A > \vec{0}$ and $J(A) > \vec{0}$. But this is precisely what rule S1 forbids.

For Step 6, note that $f \epsilon \bar{\bar{G}}_i$ and $f \epsilon G_j$ for $j \neq i$ implies $i < j$. To see this, let $g_i \epsilon G_i$, so $f << g_i << f$, and let $g_j \epsilon G_j$ be the generation for the statement containing f, so $f << g_j$. Then $g_i << f << g_j$ implies that $i < j$. This means that the use of $\bar{A}$ which replaced the original use of must follow the generation of $\bar{A}$ introduced in Step 6.

## V. THE PLANE METHOD

### A. Introduction

We now consider the application of the methods developed in Chapter 1 to our translation problem. Since the ILLIAC extended FORTRAN's DO FOR ALL is a DO SIM, by the observations of Section 2-II, we can replace rule C1 by rule S1. (In doing so, we let << be the ordering of occurrences in the given loop.) We then get the mappings $\pi$ and J to rewrite the given loop as a DO / DO FOR ALL loop as in (4). Of course, in applying rule S1 to choose $\pi$, we use the sets $< f, g >$ as calculated in Section III of this chapter.

The only additional comment we have to make is about making the optimal choice of $\pi$. Since the $\ell^j$ and $u^j$ may not be known at compile time, we do not know the values of the $M^i$ for the expressions (9) or (11) which we must try to minimize. For a practical approach, $M^i$ can be approximated in one of two ways:

(i) by a FREQUENCY statement if one appears in the program, or

(ii) using the quantities $\lambda^i$ and $u^i$ defined in Subsection IV A.

Once we have obtained $\pi$ and J, we still have the problem of rewriting the loop as a legal extended FORRRAN DO FOR ALL loop. This is always possible, but involves many practical details. Indeed, these details may introduce enough inefficiency to offset the gain due to parallel loop execution.

To prevent our being overwhelmed by details, we will restrict our algorithm to the hyperplane case. Moreover, we make two additional assumptions beyond those needed by the Hyperplane Theorem:

(i) Each $d^j$ in the given loop (13) equals $\pm 1$.

(ii) Each generated variable has no missing index variables.

## B. Writing the DO / DO FOR ALL

Assume that we have found the mapping $\pi : \mathbb{Z}^n \rightarrow \mathbb{Z}$ satisfying S1 and we have constructed the index variables $J^i$ by the algorithm of Appendix A.* Let the $I^i$ and $J^i$ be related by

(16) (a) $J^i = \sum_{j=1}^{n} h_j^i \; I^j$

(b) $I^i = \sum_{j=1}^{n} t_j^i \; J^j$

for $i = 1, \ldots, n$.

We will rewrite the given loop in the form

(17) DO $\alpha$ $J^1 = \tilde{\lambda}^1 , \tilde{u}^1$

$S = \ldots$

*This algorithm allows some choice in constructing the $J^i$. Subsection E explains how to make the best choice.

```
        DO $\alpha$ FOR ALL $(J^2, \ldots, J^n) / S$
            loop body
    $\alpha$       CONTINUE .
```

We must therefore construct the assignment statement for S and the limits $\tilde{\lambda}^1, \tilde{u}^1$. The set S wants to be assigned the value $\{ (p^2, \ldots, p^n) : (J^1, p^2, \ldots, p^n) \in \mathcal{J} \}$. This is accomplished by first writing the statement

```
    S = [ $(J^2, \ldots, J^n) / [\tilde{\lambda}^2 \ldots \tilde{u}^2]$ .CROSS. ... .CROSS. $[\tilde{\lambda}^n \ldots \tilde{u}^n]$:
X        $(\ell^1\ \Phi(.LE. , .GE. ; d^1)\ I^1)$ .AND. $(u^1\ \Phi(.GE., .LE.; d^1)\ I^1)$ ...
X                  .AND. $(u^n\ \Phi(.GE., .LE. ; d^n) \sum_j t^n_j J^j)$] .
```

Next, we substitute for each $I^i$ appearing in the statement, using Equation (16-b). (This includes any appearances of $I^i$ in the $\ell^j$ and $u^j$.) Finally, we have to choose the numbers $\tilde{\lambda}^i$, $\tilde{u}^i$ such that for every element $(p^1, \ldots, p^n) \in \mathcal{J}$, $\tilde{\lambda}^i \leq p^i \leq \tilde{u}^i$, for $i = 2, \ldots, n$.

The choice of $\tilde{\lambda}^i$, $\tilde{u}^i$ is made by the following procedure.

1. Use the algorithm of Subsection 4A for the case $k = n$, to find numbers $\lambda^j$, $u^j$ such that for every $(p^1, \ldots, p^n) \in \mathcal{J}$,

   $p^j \in [\lambda^j, \lambda^j + d^j \ldots u^j]$ , $j = 1, \ldots, n$.

   Note that this may involve rewriting the loop body. However, since it does not change any of the sets $< f, g >$, it does not affect our choice of $\pi$ and J.

2. By the theorem of Section 2, and Equation (16-a), we can let

$$\tilde{\lambda}^i = \sum_j \Phi(\lambda^j, u^j ; d^j h^i_j)\ h^i_j$$

$$\tilde{\mu}^i = \sum_j \Lambda(\lambda^j , u^j ; d^j h^i_j)\ h^i_j .$$

We also define $\tilde{\lambda}^1$, $\tilde{u}^1$ by

$$\tilde{\lambda}^1 = \sum_j \Phi(\underline{\ell}^j, \bar{u}^j ; d^1 h^1_j)\ h^1_j$$

$$\tilde{u}^1 = \sum_j \Lambda(\underline{\ell}^j , \bar{u}^j; d^1 h^1_j)\ h^1_j ,$$

where the $\underline{\ell}^j$ , $\bar{u}^j$ are defined as in Subsection 4A (for the loop as re-written in Step 1 above). Note that $\tilde{\lambda}^1$ and $\tilde{u}^1$ are loop constants, but may be unknown at compile time.

Unfortunately, we have neglected to consider the fact that the $\tilde{\lambda}^i$ could be negative, thus making invalid the above set expression defining S. In any event, it will make things easier for the compiler if all the $J^i$ vary from 1 to their upper limit. This is easily accomplished. We first change (16-a) to

(18-a) $$J^i = \sum_j h^i_j\ I^j - \tilde{\lambda}^i + 1.$$

Defining the $\iota^i$ by

$$\iota^i = \sum_j t^i_j\ (\tilde{\lambda}^j - 1) ,$$

we derive the following inverse to (18-a)

(18-b) $$I^i = \sum_j t^i_j\ J^j + \iota^i .$$

Letting $\upsilon^i = \tilde{u}^i - \tilde{\lambda}^i + 1$, we can now rewrite the DO statements as:

```
    DO α  $J^1 = 1, \upsilon^1$
    S = [ ($J^2$, ..., $J^n$) / [1 ... $\upsilon^2$] .CROSS. ... .CROSS. [1 ... $\upsilon^n$] :
X       ($\ell^1$ Φ(.LE., .GE. ; $d^1$) $I^1$) .AND. ...
X          .AND. ($u^n$ Φ(.GE., .LE. ; $d^n$) $I^n$) ]
    DO α  FOR ALL ($J^2$, ..., $J^n$) / S
```

where (18-b) is used to remove all instances of the $I^i$ from the above set expression.

We should also add an algorithm to remove redundant conditions in the set expression for S. This is easily done for some types of conditions: e.g., (2 .LE. $J^3$). However, we have not attempted to write a general algorithm for this.

## C. Reformatting the Variables

Having rewritten the DO statements using the new index variables $J^1, \ldots, J^n$, the obvious next step is to use Equation (18-b) to substitute for the $I^i$ in the loop body. However, this will not usually produce a legal DO FOR ALL loop. To illustrate this, suppose n = 3 and I and J are related by

$$\begin{aligned} I^1 &= J^1 + J^2 + 2\,J^3 & \qquad J^1 &= I^1 - I^2 - 2\,I^3 \\ I^2 &= J^2 & J^2 &= I^2 \\ I^3 &= J^3 & J^3 &= I^3 \end{aligned}$$

Then the variable occurrence $A(I^2, I^1, I^3)$ would be rewritten as $A(J^2, J^1 + J^2 + 2 J^3, J^3)$, which may not appear inside a DO FOR ALL $(J^2, J^3)$ loop.

We will solve this problem by introducing a new variable $\overline{A}$ related to A by

$$\overline{A}\,(J^1, J^2, J^3) = A(I^2, I^1, I^3) .$$

This is done as follows:

(i) Introduce the loop

```
    DO CONC β $(I^2, I^1, I^3)$ / range of A
β  $\overline{A}\,(I^1 - I^2 - 2 * I^3, I^2, I^3) = A(I^2, I^1, I^3)$
```

before the main loop.

(ii) Replace all occurrences of A by the appropriate occurrence of $\overline{A}$.

(iii) If A was a generated variable, introduce

```
    DO CONC γ $(I^2, I^1, I^3)$ / range of A
γ  $A(I^2, I^1, I^3) = \overline{A}(I^1 - I^2 - 2 * I^3, I^2, I^3)$
```

after the main loop.

Subsection E will consider the problem of rewriting these DO CONC loops as legal DO FOR ALL loops.

We note here that the only time a legal DO FOR ALL will be formed by simply substituting for the $I^j$ inside the loop body is when $\pi$ is a coordinate projection - i.e., when $J^1 = I^k$ for some k. In this case, each $J^i$ may be chosen to be one of the $I^j$, so the rewriting of the loop body can be done by the procedures of the Coordinate Algorithm. For the rest of our discussion, we assume that this is not the case.

We now generalize the method we applied above for the variable A to the following algorithm for reformatting* variables.

1. For notational convenience, assume that each occurrence f of a variable VAR is of the form

   $$\text{VAR}(I^{j_1} + f^{j_1}, \ldots, I^{j_k} + f^{j_k}, f_c^1, \ldots, f_c^m),$$

   where $j_1 < \ldots < j_k$, and the $f^{j_i}$, $f_c^i$ are loop constants. (Of course, k and m depend upon the variable but not upon the particular occurrence of that variable.)

2. For each non-scalar variable VAR appearing in the loop such that $0 < k < n$:**

   (a) Introduce a new $(n + m)$ - dimensional variable VARX, whose jth dimension is defined to be equal to:

      (i) The ith dimension of VAR, if $j = j_i$.

      (ii) The $(j - n + k)$th dimension of VAR if $j > n$.

*Do not confuse reformatting with the reallocation done in Step 13 of the Coordinate Algorithm of Subsection IV C.

**Here, and in Step 3, k and m are as defined in Step 1, for the particular variable under discussion.

(iii) $|u^j - \lambda^j| + 1$ otherwise, where $u^j$, $\lambda^j$ are the numbers found in Subsection B (in Step 1 of the procedure for choosing the $\tilde{\lambda}^i$, $\tilde{u}^i$).

(b)- Insert the following loop before the main loop:

DO β CONC FOR $(I^1, \ldots, I^{n+m})$ / range of VARX

β VARX $(\xi^1, \ldots, \xi^{n+m})$ = VAR $(I^{j_1}, \ldots, I^{j_k}, I^{n+1}, \ldots, I^{n+m})$,

$$\text{where } \xi^j = \begin{cases} I^j - \Phi(\lambda^j, u^j; d^j) + 1 \text{ if } j = j_i \text{ for some } i \\ \\ I^j \text{ otherwise.} \end{cases}$$

(c) Replace each occurrence f of VAR in the loop by VARX$(e^1, \ldots, e^n, f_c^1, \ldots, f_c^m)$, where

$$e^j = \begin{cases} I^j + f^j \text{ if } j = j_i \text{ for some } i \\ \\ I^j \text{ otherwise.} \end{cases}$$

3. For each non-scalar variable VAR appearing in the loop (as rewritten in Step 2) such that $k = n$:

   (a) Introduce a new $(n + m)$-dimensional variable $\overline{\text{VAR}}$. If the dimensions of VAR are $\delta^1, \ldots, \delta^{n+m}$ then the dimensions $\tilde{\delta}^1, \ldots, \tilde{\delta}^{n+m}$ are defined as follows.

For i = 1, ..., n; let

$$\underline{\delta}^i = \sum_j \Phi(1, \delta^j; h^i_j)\ h^i_j$$

$$\overline{\delta}^i = \sum_j \Lambda(1, \delta^j; h^i_j)\ h^i_j\ .$$

$$\text{Then } \tilde{\delta}^i = \begin{cases} \overline{\delta}^i - \underline{\delta}^i + 1 & \text{if } i \leq n \\ \delta^i & \text{if } n < i \leq n + m\ . \end{cases}$$

(b) Insert the following loop before the main loop:

DO β CONC FOR $(I^1, \ldots, I^{n+m})$ / range of VAR

β $\overline{\text{VAR}}\ (\sum_j h^1_j I^j - \underline{\delta}^1 + 1, \ldots, \sum_j h^n_j I^j - \underline{\delta}^n + 1,$
$I^{n+1}, \ldots, I^{n+m}) = \text{VAR}(I^1, \ldots, I^{n+m})\ .$

(c) Replace each occurrence f of VAR in the loop

by

$$\overline{\text{VAR}}\ (J^1 + \tilde{\lambda}^1 - \underline{\delta}^1 + \sum_j h^1_j f^j, \ldots,$$
$$J^n + \tilde{\lambda}^n - \underline{\delta}^n + \sum_j h^n_j f^j, f^1_c, \ldots, f^m_c).$$

(d) If VAR was a generated variable, add the following loop after the main loop:

DO β CONC FOR $(I^1, \ldots, I^{n+m})$ / range of VAR

β VAR $(I^1, \ldots, I^{n+m}) =$
$\overline{\text{VAR}}\ (\sum_j h^1_j I^j - \underline{\delta}^1 + 1, \ldots, \sum_j h^n_j I^j - \underline{\delta}^n + 1,$
$I^{n+1}, \ldots I^m)\ .$

D. Discussion of Reformatting

We begin the discussion with an explanation of the preceding algorithm.

1. This definition of the $f^j$ and $f_c^j$ for the occurrence f is used in steps 2 and 3, as is the definition of k and m for the variable of that occurrence. It is well to remember that the loop constants $f_c^j$ may be functions of an index variable of some outer loop containing the given loop (13).

2. If any of the $I^j$ are missing from the occurrences of VAR, it is necessary to put them in so that we may apply Step 3. This is primarily because the DO FOR ALL syntax requires all occurrences to involve the entire multi-index $(J^2, \ldots, J^n)$. Also, it will be impossible to represent $(I^{j_i}, \ldots, I^{j_k})$ in terms of k of the $J^i$.

   Note that the DO CONC loop of (b) is easy to translate into DO / DO FOR ALL loops.

   The remarks we make for Step 3 about the last m dimensions of VAR apply to this step as well.

3. Here, we replace VAR by the variable $\overline{VAR}$ which is related to it by

$$(11\text{-}9) \qquad \overline{VAR}(J^1, \ldots, J^n, I^{n+1}, \ldots, I^{n+m}) = VAR(I^1, \ldots, I^{n+m}).$$

The correctness of the dimensions $\tilde{\delta}^i$ follows from the theorem of Section II. I.e., the subscript values for the occurrence of $\overline{VAR}$ in (b) range from 1 through the $\tilde{\delta}^i$.

Note that we have reformatted the entire (n+m)-dimensional array VAR, even though the loop may only reference one n-dimensional slice, and can reference at most one slice for each occurrence of VAR in the loop. To see why this may be necessary, consider the two occurrences $A(I^1, I^2, K)$ and $A(I^1, I^2, L)$, L and K unknown loop constants. Since these two occurrences may reference the same slice, or different slices, we must reformat all of A. However, if all occurrences of A are of the form A( ..., ..., K), then we need only reformat the single $I^3 = K$ slice.

The generalization of our algorithm to one which only reformats the part of the array that may be referenced is straightforward, but is tedious to write in full detail. Note, however, that it is influenced by the placement of the reformatting loop, which is discussed below.

The efficient translation of the DO CONCs produced by (b) and (d) may be difficult. It will be discussed at length in subsection D.

Note that 3(b) need not be done if all uses of VAR reference only values which are generated in the loop.

Since reformatting must be done for essentially all non-scalar variables in the loop, it represents a large "overhead cost" for the Hyperplane Method. However, the loop (13) will often be contained in a larger loop (either a programmed loop, or another DO loop). The reformatting can then be moved outside the loop. Any occurrences of the variable VAR after the reformatting loop of Step 3(b), and before that of Step 3(d) can be replaced by occurrences of $\overline{\text{VAR}}$. The required occurrence mappings are easily derived from the formula in Step 3(b).

The problem of finding the optimal location for the reformatting loops is very similar to the problems encountered in compiler optimization.

## E. Translating the DO CONCs

We now consider the problem of translating the DO CONCs produced in Steps 3(b) and (d) above. Since a loop involving a DO FOR ALL will be much more efficient than an ordinary sequential DO loop, we will almost always want to use a DO FOR ALL. However, consider the following loop which might be produced in Step 3(b);

(19) DO $\beta$ CONC $(I^1, I^2)$ / range of A

$\beta$ $\overline{A}(2 * I^1 + 3 * I^2 - 2, I^1 + I^2) = A(I^1, I^2)$ .

Because of the complicated subscripting of $\overline{A}$, this cannot be translated into a legal DO FOR ALL on $I^1$ or $I^2$ or $(I^1, I^2)$.

In such a case, the reformatting is done in stages as follows:

Let $J_0^i$, $J_1^i$, ..., $J_\tau^i$ be the sequence of variables constructed in Appendix A, where $J_0^i = I^i$ and $J_\tau^i = J^i$. The DO CONC loop generated in Step 3(b) of subsection B is replaced by a sequence of the following loops, for $\sigma = 0, \ldots, \tau - 1$:

(20) DO β CONC FOR $(J_\sigma^1, \ldots, J_\sigma^n)$ / range of $VAR_\sigma$

B $VAR_{\sigma+1}(J_{\sigma+1}^1 - \underline{\delta}_{\sigma+1}^1 + 1, \ldots, J_{\sigma+1}^n - \underline{\delta}_{\sigma+1} + 1) =$

$VAR_{\sigma+1}(J_\sigma^1, \ldots, J_\sigma^n)$

where the $\underline{\delta}_{\sigma+1}^i$ are computed as in Step 3(a), and the $J_{\sigma+1}^i$ are replaced by their values given in Equation (A-2) of Appendix A.*
The $VAR_\sigma$ are new variables which we have introduced.

From Equation (A-2), we see that the DO CONC loop (20) can always be rewritten with a DO FOR ALL $J_\sigma^p$. Thus, if there is more than one choice for the index p, we should choose the one for which $J_\sigma^p$ is the best DO FOR ALL variable. We can add this to the algorithm of Appendix A for choosing J.

Suppose that for some particular σ and some r, the algorithm of Appendix A yields $q_r^\sigma = 0$. This then implies that $q_r^p = 0$ for all $p > \sigma$, and $J^r = J_\tau^r = J_\sigma^r$ (neglecting the permutation mentioned in the footnote). Hence, we can replace $J_{\sigma+1}$ by $J_\tau$ (and $VAR_{\sigma+1}$ by $VAR_\tau$) in (20) and get a DO CONC which can be translated into a DO FOR ALL

*Recall that for $\sigma + 1 = \tau$, so $J_{\sigma+1}^i = J^i$, an additional permutation of the superscripts may be required.

$J^r_\sigma$ loop. This eliminates the last $\tau - (\sigma + 1)$ stages of the reformatting. For convenience of notation, set $\tau = \sigma + 1$ in this case.

We thus have a sequence of DO CONC loops (20) for the variable VAR. The construction of this sequence is essentially independent of the variable. I.e., the process is really the construction of the $J^i_\sigma$. Hence, it is only done once for all the variables.

The dimensions of the array $VAR_0$ are the same as those of VAR. The dimensions of $VAR_1, \ldots, VAR_\tau$ are determined sequentially by the same method used in Step 3(a) of Subsection C. The dimensions of $VAR_\tau$ will then equal those of $\overline{VAR}$.

The storage allocations for the $VAR_\sigma$ are made so as to permit the DO FOR ALL in the translation of (20). If the storage allocation for VAR agrees or can be made to agree with the allocation of $VAR_0$, then VAR is substituted for $VAR_0$ in (20). Otherwise, the assignment statement

$$VAR_0 = VAR$$

must precede the reformatting. Similarly, the allocation of $\overline{VAR}$ must allow the DO FOR ALL on $(J^2, \ldots, J^n)$. If this is consistent with the storage allocation required by $VAR_\tau$, then replace $VAR_\tau$ with $\overline{VAR}$. Otherwise, the statement

$$\overline{VAR} = VAR_\tau$$

must follow the loops (20).

The translation of the DO CONC loops generated by Step 3(d) of Subsection B is obtained by the obvious reversal of the loops constructed above.

The above procedure sounds very long and costly when described abstractly. Observe, though, that $\tau = 1$ corresponds to the case in which the DO CONC loops of 3(b) and (d) may be written immediately with a DO FOR ALL (except, perhaps, for storage reallocation). In general, as mentioned in Appendix A, we have $\tau \leq \text{minimum} \ \{ \ |h_i| : h_i \neq 0 \ \}$. In practice, $\tau$ will usually be small, and will often equal 1.

To illustrate this procedure, consider the DO CONC of (19) which was obtained from the $\pi$ defined by $\pi\,[(I^1, I^2)] = 2\,I^1 + 3\,I^2$. Application of the algorithm of Appendix A gives

$I^1 = J_0^1, J_1^1, J_2^1 = J^1$, where

$$\begin{array}{ll} J_1^1 = I^1 + I^2 & J^1 = J_1^2 + 2\,J_1^1 \\ J_1^2 = I^2 & J^2 = J_1^1 \end{array}$$

If the dimensions of A are 20, 30, then we get from 3(a)

$$\begin{array}{lll} \underline{\delta}_1^1 = 2 & \bar{\delta}_1^1 = 50 & \tilde{\delta}_1^1 = 49 \\ \underline{\delta}_1^2 = 1 & \bar{\delta}_1^2 = 30 & \tilde{\delta}_1^2 = 30 \\ \underline{\delta}_2^1 = \underline{\delta}^1 = 3 & \bar{\delta}^1 = 128 & \tilde{\delta}^1 = 126 \\ \underline{\delta}^2 = 1 & \bar{\delta}^2 = 49 & \tilde{\delta}^2 = 49 \end{array}$$

Thus, 49, 30 are the dimensions of $A_1$, and 126, 49 are the dimensions of $A_2 = \bar{A}$. (We assume the default allocation for A and $\bar{A}$, so no reallocation is necessary.) We can then translate (19) into:

```
          DO $\beta_1$ $I^2$ = 1, 30
          DO $\beta_1$ FOR ALL $I^1$ / [1 ... 20]
$\beta_1$      $A_1(I^1 + I^2 - 1, I^2) = A(I^1, I^2)$
          DO $\beta_2$ $J_1^2$ = 1, 30
          DO $\beta_2$ FOR ALL $J_1^1$ / [1 ... 49]
$\beta_2$      $\bar{A}(J_1^2 + 2 * J_1^1, J_1^1) = A_1(J_1^1, J_1^2)$
```

Observe that these loops require a total of 60 ILLIAC iterations, compared to 600 iterations if (19) were executed sequentially. A 49 by 30 temporary array $A_1$ had to be introduced. However, note that $A_1$ can be overlapped with any other similarly introduced temporary array - i.e., with $VAR_1$ for any variable VAR.

## F. Avoiding Reformatting

There is one case in which reformatting can be avoided altogether, namely the case in which the following hold:

(i) $\pi$ satisfies rule C1, so the loop (13) can be rewritten with a DO CONC.

(ii) $a_i = 0$ for some i, where $a_i$ is as in (8) of Section I-7.

An examination of the algorithm of Appendix A shows that (ii) implies that $J^i = I^i$, and in (6-b) and (8-b) we have $t^i_j = 0$ if $j \neq i$.* Therefore, using (8-b) to substitute for the $I^j$ in the loop body produces a valid DO FOR ALL $J^i$ loop body. Thus, we need only translate the DO CONC ($J^2, \ldots, J^n$) / S statements into a sequence of DOs followed by a DO FOR ALL $J^i$. This translation procedure is similar to the ones we have already performed, so a detailed algorithm is omitted.

If $a_{i_1} = \ldots = a_{i_k} = 0$, the above procedure can be generalized to yield a DO FOR ALL ($J^{i_1}, \ldots, J^{i_k}$) loop.

## G. <u>The Number of ILLIAC Iterations</u>

The mapping $\pi$ should really be chosen to minimize the total number of ILLIAC iterations. We can now write a formula for that number.

Let $\lceil a \rceil$ denote the smallest integer greater than or equal to a. From Subsection B, it is easy to see that the number of ILLIAC iterations needed to execute the loop is

$$(\upsilon^1 + 1) \lceil N \rceil ,$$

where $N = (\upsilon^2 + 1) \ldots (\upsilon^n + 1) / 64$ (assuming a 64 P.E. ILLIAC).

*We are again neglecting the first superscript permutation in the algorithm of Appendix A. The necessary modifications to the discussion are obvious. Note that $J^i$ will never be $J^1$.

The expression (9) of Section 1-VII which we decided to minimize is actually equal to $\upsilon^1$. Since computation of $\upsilon^2, \ldots, \upsilon^n$ involved the variables $J^i$, it is clear that the $\upsilon^i$ cannot be written as a simple function of $\pi$ when $i > 1$.

## H. An Example

It is of interest to see just what sort of loop if produced by these procedures. We consider the following simple relaxation loop:

```
      DO 77 I1 = 2, N
      DO 77 I2 = 3, M
      A(I1, I2) = .25 * ( A(I1 + 1, I2) + A(I1 - 1, I2)
                + A(I1, I2 + 1) + A(I1, I2 -1) )
77    CONTINUE
```

with A defined by

```
      DIMENSION  A(35, 50) .
```

Application of the method used in the proof of the Hyperplane Theorem gives the optimal mapping $\pi : \mathbb{Z}^2 \to \mathbb{Z}$ defined by

$$\pi[(I^1, I^2)] = I^1 + I^2 .$$

The algorithm of Appendix A, with the addition made in Subsection E, gives

$$J^1 = I^1 + I^2 \qquad I^1 = J^2$$

$$J^2 = I^1 \qquad I^2 = J^1 - J^2$$

We now apply the algorithm of Subsection B to write the DO and DO FOR ALL statements. This requires first applying the algorithm of Subsection IV A for the case $k = n = 2$, to find the $\lambda^i$, $u^i$. Applying that algorithm gives the following results :

$$\lambda^1 = 1 \qquad u^1 = 35$$
$$\lambda^2 = 1 \qquad u^2 = 50$$

and rewrites the loop as

```
      DO 77 I^1 = 1, N - 1
      DO 77 I^2 = 1, M - 2
      A(I^1 + 1, I^2 + 2) = .25 * (A(I^1 + 2, I^2 + 2)
               + A(I^1, I^2 + 2) + A(I^1 + 1, I^2 + 3)
               + A(I^1 + 1, I^2 + 1) )
77    CONTINUE .
```

Continuing the algorithm of Subsection B, we get

$$\tilde{\lambda}^1 = 2 \qquad \tilde{u}^1 = N+M-3 \qquad \iota^1 = 0 \qquad \upsilon^1 = N+M-4$$
$$\tilde{\lambda}^2 = 1 \qquad \tilde{u}^2 = 35 \qquad \iota^2 = 1 \qquad \upsilon^2 = 35$$

and (18-a, b) become

$$J^1 = I^1 + I^2 - 1 \qquad I^1 = J^2$$
$$J^2 = I^1 \qquad I^2 = J^1 - J^2 + 1$$

We then get the following loop control statements:

```
DO  77 $J^1$ = 1, N + M - 4
S = [ $J^2$ / [1 ... 35] : (1 .LE. $J^2$) .AND.
    ((N - 1) .GE. $J^2$) .AND. (1 .LE. ($J^1$ - $J^2$ + 1))
    .AND. ((M - 2) .GE. ($J^1$ - $J^2$ + 1)) ]
DO 77 FOR ALL $J^2$ / S .
```

We now apply the algorithm of Subsection C to the variable A. We then get

$\underline{\delta}^1 = 2 \qquad \bar{\delta}^1 = 85 \qquad \tilde{\delta}^1 = 84$

$\underline{\delta}^2 = 1 \qquad \bar{\delta}^2 = 35 \qquad \tilde{\delta}^1 = 35$ .

We have the following DO CONC loops inserted before and after the main loop, respectively:

```
        DO 177 CONC FOR ($I^1$, $I^2$) / [1...35] .CROSS. [1...50]
177     $\bar{A}$ ($I^1 + I^2 - 1, I^1$) = A($I^1, I^2$)
        DO 277 CONC FOR ($I^1$, $I^2$) / [1...35] .CROSS. [1...50]
277     A($I^1, I^2$) = $\bar{A}$($I^1 + I^2 - 1, I^1$) .
```

The body of the loop is rewritten as follows:

$$\bar{A}(J^1 + 3, J^2 + 1) = .25 * ( \bar{A}(J^1 + 4, J^2 + 2) + \bar{A}(J^1 + 2, J^2) + \bar{A}(J^1 + 4, J^2 + 1) + \bar{A}(J^1 + 2, J^2 + 1) ).$$

The DO CONC loops may be immediately translated into DO $I^1$ / DO FOR ALL $I^2$ loops. Combining all of this, we get the following rewriting of the loop:

```
       DO 177 $I^1$ = 1, 35
       DO 177 FOR ALL $I^2$ / [1 ... 50]
177    $\bar{A}(I^1 + I^2 - 1, I^1) = A(I^1, I^2)$
       DO  77 $J^1$ = 1, N + M - 4
       S = [ $J^2$ / [1 ... 35] : (1 .LE. $J^2$) .AND. ((N - 1) .GE. $J^2$)
           .AND. (1 .LE. ($J^1 - J^2$ + 1)) .AND. ((M - 2) .GE.
           ($J^1 - J^2$ + 1)) ]
       DO 77 FOR ALL $J^2$ / S
       $\bar{A}(J^1 + 3, J^2 + 1) = .25 * ( \bar{A}(J^1 + 4, J^2 + 2) + \bar{A}(J^1 + 2, J^2)$
                            $+ \bar{A}(J^1 + 4, J^2 + 1) + \bar{A}(J^1 + 2, J^2 + 1) )$
77     CONTINUE
       DO 277 $I^1$ = 1, 35
       DO 277 FOR ALL $I^2$ / [1 ... 50]
277    $A(I^1, I^2) = \bar{A}(I^1 + I^2 - 1, I^1)$
```

Except for the redundant condition (1 .LE. $J^2$), this is about as efficient a rewriting of the loop as we can expect. As mentioned in Subsection C, an algorithm to remove such conditions can be constructed.

Despite its complexity, the above loop will probably run about 6 times faster on the ILLIAC than the original loop. Furthermore, most such relaxation loops occur inside another loop which includes a convergence test. The reformatting loops could be placed outside this outer loop. We would then get a loop which runs about 9 times faster than the original.

## I. Simultaneous Application of the Method

The Hyperplane Method requires reformatting all the variables in the loop, which introduces a large overhead cost. One way to reduce this overhead is to use the same reformatted variables in more than one loop.

Suppose that there are several loops in the program which satisfy the hypothesis of the Hyperplane Theorem, and are all of the same dimension. (I.e., they have a common value of n.) If the same index variables $J^1, \ldots, J^n$ are used for all the loops then it may be possible to do the reformatting only once.

In fact, such a single n-tuple of index variables can be chosen which allows all the loops to be written with a DO FOR ALL $(J^1, \ldots, J^n)$. All we do is apply our method of choosing $\pi$ to the sets $< f, g >$ of all the loops taken together. Any $\pi$ satisfying rule S1 will then work for all the loops. To find the optimal choice, assign to each loop a weighting factor proportional to both the execution time of the loop body and the frequency of execution of the entire loop. Then $\pi$ is chosen to minimize the expression formed as follows: For each loop, take the expression (a) of Section -VII and multiply it by the above wieghting factor. Then, sum over all the loops.

# CHAPTER FIVE

# LOOP HISTORY SIMULATION TO DERIVE CONCURRENCY

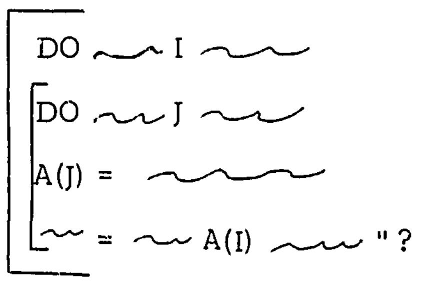
## I. INTRODUCTION

We have derived methods for determining sets of points, in the control index space of a nest of FORTRAN DO loops, for which concurrent and ILLIAC-simultaneous execution of the statements of the loop body can be performed. These analysis methods rely on the subscript expressions of array references being loop-constant affine transformations of the loop index variables of a restricted form (see Chapter 3 - Section I). In cases where these restrictions are not met, sets of loop index points for concurrent loop body execution can be derived by a simulation of the control history of the loop. The simulation method works best for nests of DO loops in which the control limit parameters (at least at the outermost level) are known constants. The method also depends on the array reference subscript transformations being dependent only on quantities determined within the loop or else determinable statically from the rest of the program.

## II. FIRST EXAMPLE

The following discussion shows an example of a very simple loop form which does not satisfy the requirements for the analytic methods of parallelism detection. The simulation technique is applied to the example and to a generalization of it, showing results in determining an improvement in execution speeds derived from potential simultaneous executions of previously separate executions of the loop body.

The general problem of this example can be stated as: "What kind of simultaneity or concurrent execution can be found in loops of the form:



Depending on the forms of the subscripts, the generating statement may access one cell of the array more than once (particularly if there are two generators, one in I and one in J and the set of I and J values has a non-empty intersection). Moreover, the generation-use or overwrite relations between the generation and use statements shift as the loop indices change values. (This is in marked contrast to the simple singly-subscript linear forms in which the relations are constant over the whole history of the loop.)

Consider the following example, particularly with view to changing the inner or outer loop, to a DO FOR ALL loop. It is presented in terms of its running values when written in each form.

Original Loop (sequential)

```
        DO 1 J = 1, 3
        DO 2 I = 1, 3
        A(J) = B(J) + I
        C(J) = A(I)
2       CONTINUE
1       CONTINUE
```

Proposed Transformed Loop (simultaneous execution of previously "outer" loop)

```
        DO 2 I = 1, 3
        DO 1 FOR ALL J/[1, 2, 3]
        A(J) = B(J) + I
        C(J) = A(I)
1       CONTINUE
2       CONTINUE
```

History of loop computation in both forms.

| I,J Value | Statement Executed | I,J Value | Statement Executed |
|---|---|---|---|
| 1,1 | A(1) = B(1) + 1 | 1,1 | A(1) = B(1) + 1 |
|  | C(1) = A(1) ≡ B(1) + 1 | 1,2 | A(2) = B(2) + 1 |
| 2,1 | A(1) = B(1) + 2 | 1,3 | A(3) = B(3) + 1 |
|  | C(1) = Initial value of A(2) | 1,1 | C(1) = A(1) ≡ B(1) + 1 |
| 3,1 | A(1) = B(1) + 3 | 1,2 | C(2) = A(1) ≡ B(1) + 1 |
|  | C(1) = Initial value of A(3) | 1,3 | C(3) = A(1) ≡ B(1) + 1 |
| 1,2 | A(2) = B(2) + 1 | 2,1 | A(1) = B(1) + 2 |
|  | C(2) = A(1) ≡ B(1) + 3 | 2,2 | A(2) = B(2) + 2 |
| 2,2 | A(2) = B(2) + 2 | 2,3 | A(3) = B(3) + 2 |

| I,J Value | Statement Executed |
|---|---|
| | C(2) = A(2) ≡ B(2) + 2 |
| 3,2 | A(2) = B(2) + 3 |
| | C(2) = Initial value of A(3) |
| 1,3 | A(3) = B(3) + 1 |
| | C(3) = A(1) ≡ B(1) + 3 |
| 2,3 | A(3) = B(3) + 1 |
| | C(3) = A(2) ≡ B(2) + 3 |
| 3,3 | A(3) = B(3) + 3 |
| | C(3) = A(3) ≡ B(3) + 3 |

| I,J Value | Statement Executed |
|---|---|
| 2,1 | C(1) = A(2) ≡ B(2) + 2 |
| 2,2 | C(2) = A(2) ≡ B(2) + 2 |
| 2,3 | C(3) = A(2) ≡ B(2) + 2 |
| - - - - - - - - | - - - - - - - - - - - - |
| etc. | |

In the simultaneous J form, the horizontal dotted lines separate triples of statements which are executed simultaneously. That is, all three "A(J) = ---" statements are executed simultaneously for J = 1, J = 2, and J = 3. The syntax of the "DO FOR ALL" loop indicates that J is the "simultaneous" index and takes all the integral values 1, 2, and 3 during the execution of each statement. Each value is assigned to a different processor.

The proposed rewriting does not compute the same values as the original. In particular, the final C(1) and C(2) values should be the same as the initial A(3) value, but will be B(3) + 3.

In terms of the sequential loop

- the generation of A(1) at (3, 1) (I = 3, J = 1) provides the value for the use of A(1) at (1, 2) across index set points.

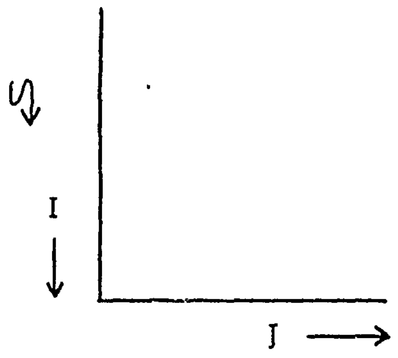
- the generation of A(2) at (2, 2) provides the value for the use of A(2) still within the same loop index point.
- the use of A(3) at (3, 2) is the initial value of A(3) and so the generations of A(3) at (1, 3), (2, 3) and (3, 3) must not precede this use.

These three situations are <u>not all</u> satisfied by the proposed rewriting.

Even though this simple rewriting was impossible, that does not mean that there is <u>no</u> inherent concurrency in the example.

Consider the sequential loop example expanded to I = 1, 5 and J = 1, 5 but with no other changes.

The notation of the following matrix represents a history of simulation of the loop body in the order in which the loop indices are incremented:



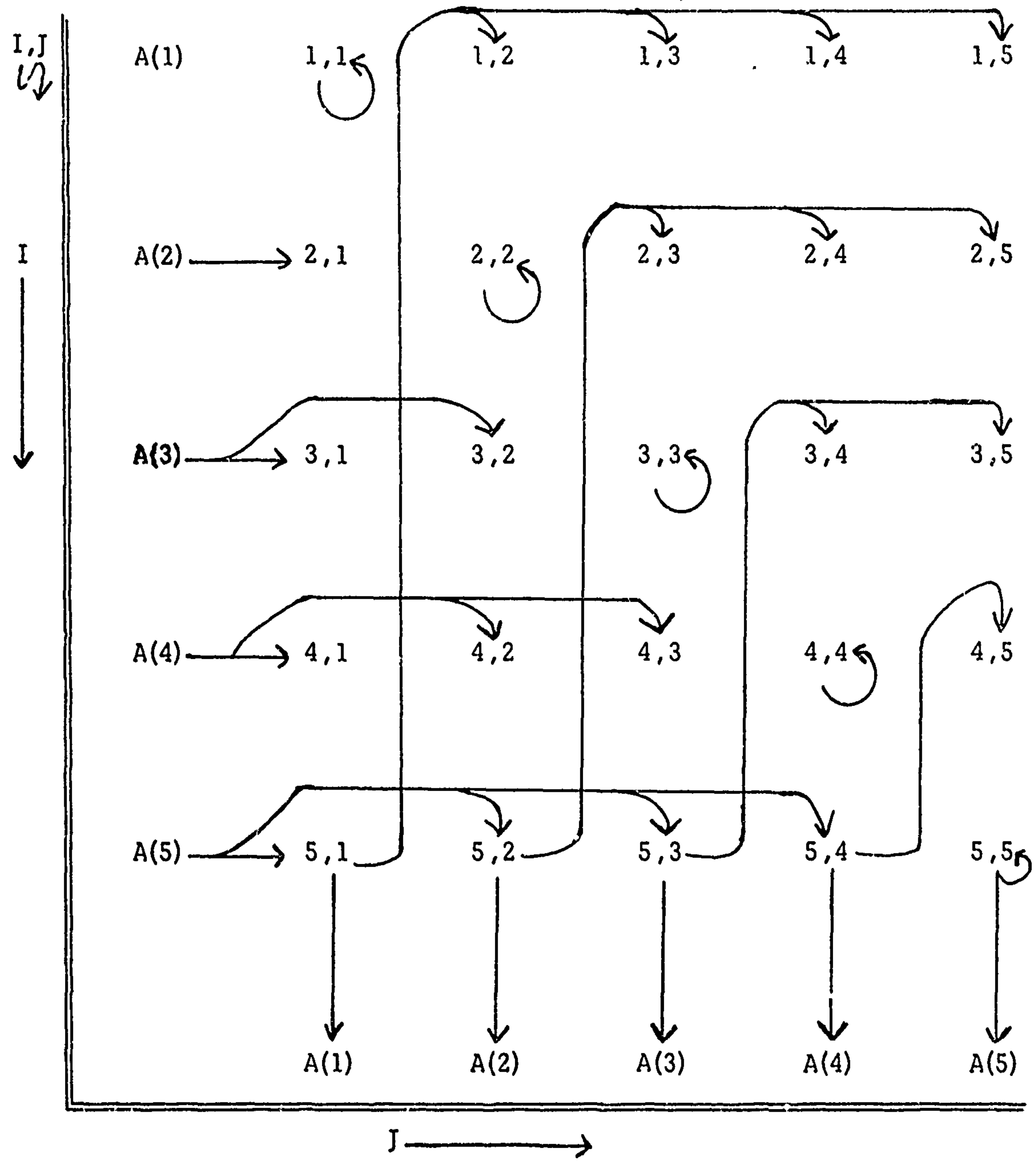
means that the I loop is iterated before the J loop (the I loop is interior to the J loop).

The ordered pairs of values in the matrix stand for I, J pairs at a point in time.

The arrows demonstrate generation - use relations in the simulation. They are derived by inspecting subscript forms on the generation and use statements keeping in mind that a generator precedes in time a use of that value. Thus the arrows point from generations to uses. This inspection procedure could easily be mechanized.

The left hand margin nodes "A(1)", "A(2)" etc. are the initial values of A on entry to the loop. The lower margin nodes "A(1)", "A(2)" etc. are final values of A on exit from the loop.

Generations and uses of the variable A for 5 x 5 case derived from the example:


I,J
I
A(1)
A(2)
A(3)
A(4)
A(5)
1,1
1,2
1,3
1,4
1,5
2,1
2,2
2,3
2,4
2,5
3,1
3,2
3,3
3,4
3,5
4,1
4,2
4,3
4,4
4,5
5,1
5,2
5,3
5,4
5,5
A(1)
A(2)
A(3)
A(4)
A(5)
J

Since the array B is used only as an input to the loop (only in "uses") and the array C is used only as output of the loop (only in "generations") the essential ordering information lies only within the uses and generations of A.

To the use-generation arcs can be added a set of arcs for "overwrite avoidance precedence". In this example there is in general more than one generating instance of a value. (The gen-use arcs link "latest" generation with "proper" use.) For instance, I, J points (1, 1), (2, 1), (3, 1), (4, 1) and (5, 1) all generate a value for A(1). Note that (1, 1) ... (4, 1) all precede (5, 1) in time and that (5, 1) generates a value which will be used by (1, 2), (1, 3), (1, 4) and (1, 5) and will be the output value for A(1).

To prevent overwrite of the A(1) value properly generated by (5, 1) arcs must be drawn from (1, 1) to (5, 1), (2, 1) to (5, 1), (3, 1) to (5, 1) and (4, 1) to (5, 1). This process, too, can be mechanized.

This set of arcs and nodes can be used to study the sets of nodes (or values of I, J) for which the body of the loop can be executed concurrently. An incidence matrix is formed and used according to the method of Ramamoorthy [ 6, particularly pages 5-7 ].

In the partition list notation:

In is an input value node

On is an output value node

nn is a point in the history of the loop where

$n_1 n_2$ stands for $I = n_1$, $J = n_2$ .

In the list of arcs notation:

the arcs are labelled according to the pair of nodes with head of arc preceding tail.

The set of arcs used to form the incidence matrix is:

| | | |
|---|---|---|
| 11, 51 | 41, 51 | I2, 21 |
| 12, 52 | 42, 52 | I3, 31 |
| 13, 53 | 43, 53 | I3, 32 |
| 14, 54 | 44, 54 | I4, 41 |
| 15, 55 | 45, 55 | I4, 42 |
| 21, 51 | 51, 12 | I4, 43 |
| 22, 52 | 51, 13 | I5, 51 |
| 23, 53 | 51, 14 | I5, 52 |
| 24, 54 | 51, 15 | I5, 53 |
| 25, 55 | 52, 23 | I5, 54 |
| 31, 51 | 52, 24 | 51, O1 |
| 32, 52 | 52, 25 | 52, O2 |
| 33, 53 | 53, 34 | 53, O3 |
| 34, 54 | 53, 35 | 54, O4 |
| 35, 55 | 54, 45 | 55, O5 |

Note that I1 never appears because the initial value of A(1) is never used. It is reasonable to add an assumption to the results of the method: that all the inputs precede any output. This is not strictly necessary in this case. Also note that "self-loops" like {11, 11}, although indicating that A(1) is generated and then used <u>within</u> the point (1,1), are not included in the incidence matrix. The aim is to derive sets of distinct points for which the loop body can be executed in parallel. For this 5 x 5 case the technique described by Ramamoorthy gives concurrency partitions ("<u>earliest</u> execution initiation") as:

| <u>time</u> | <u>Set</u> |
|---|---|
| $t_1$ | (I1, I2, I3, I4, I5), 11, 22, 33, 44 |
| $t_2$ | 21, 31, 32, 41, 42  43 |
| $t_3$ | 51 |
| $t_4$ | (O1), 12, 13, 14, 15 |
| $t_5$ | 52 |
| $t_6$ | (O2), 23, 24, 25 |
| $t_7$ | 53 |
| $t_8$ | (O3), 34, 35 |
| $t_9$ | 54 |
| $t_{10}$ | (O4), 45 |
| $t_{11}$ | 55 |
| ($t_{12}$ | (O5)   ) |

The I and O events are "outside" the actual loop. One can assume <u>all</u> I's are done at entry and can assume there won't be any use of an O until the exit. Note that:

1) the $t_1$ and $t_2$ (non-I stuff) partitions can be merged, and

2) $t_{12}$ really is an external time slice.

Thus the whole loop history is performable in 10 chunks instead of 25.

The original loop for Max { [I, J] } = [5, 5] can now be rewritten to reflect the new time slices:

```
        DO  1  FOR ALL J / [1, 2, 3, 4]          }
        A(J) = B(J) + J                          }  t1
1       C(J) = A(J)                              }
*       " DO  2  FOR ALL I / [2, 3, 4] "         }
**      " DO  2  FOR ALL J / [1, 2, ..., I-1]"   }  t2
        A(J) = B(J) + 1                          }
2       C(J) = A(I)                              }
        DO 4 I = 1, 4                            }
        A(I) = B(I) + 5                          }
        C(I) = A(5)                              }
        DO  3  FOR ALL J / [I+1, I+2, ..., 5]    }  t3 - t10
        A(J) = B(J) + I                          }
3       C(J) = A(I)                              }
4       CONTINUE                                 }
        A(5) = B(5) + 5                          }  t11
        C(5) = A(5)                              }
```

---

* This "DO FOR ALL I" is not valid because I is not a proper "simultaneous" variable inside the body: it does not appear on the left hand side as a subscript.

**This "DO FOR ALL J / [∼∼∼, I - 1]" is probably not syntactically valid.

Note: The problems with syntax or semantics at $t_2$ result from the values of A(1) - A(3) and C(1) - C(3) being over-written below. There is a generalization of this case (for N > 2) which shows just how much improvement in execution time may be possible. Ordinary Fortran:

```
      DO 1 J = 1, N
      DO 2 I = 1, N
      A(J) = B(J) + I          use of both J and I in
                               gen. of A(J)
      C(J) = A(I)              Use of A(I)
2     CONTINUE
1     CONTINUE
```

requires $N^2$ executions of body.

Rewritten:

```
N-1 parallel executions          DO 1 FOR ALL J / [1, 2, ..., N-1]   Note
equiv. to 1 execution if         A(J) = B(J) + J                     change of
≧ N - 1 processors             1 C(J) = A(J)                         variables
available

(N-2) (N - 3) parallel           DO 2 I = 2, N - 1
      2       executions         DO 2 FOR ALL J / [1, 2, ..., I-1]
equiv. to N-2 execu-             A(J) = B(J) + I
tions if ≧ N - 2 proc-         2 C(J) = A(I)
essors available

N - 1 executions                 DO 4 I = 1, N - 1          Note change of
                                 A(I) = B(I) + N            variables and intro-
                                 C(I) = A(N)                duction of a constant N.

(N-1) (N-2) parallel             DO 3 FOR ALL J / [I + 1, I + 2, ..., N]
      2
executions equiv. to             A(J) = B(J) + I
N-1 executions if ≧            3 C(J) = A(I)
N-2 processors
available
                               4 CONTINUE

1 execution                      A(N) = B(N) + N.   This could have been in
                                                    "DO 4 I" loop if "DO 3
                                 C(N) = A(N)        FOR ALL J" not executed
                                                    at least once.
```

Total equivalent executions of "loop body" = 3N - 2.

## III. SECOND EXAMPLE

Another example of the use of a simulation technique demonstrates that there is often a considerable difference between the amount of concurrent execution that can be expressed with a "linearized" syntactic form such as a DO FOR ALL and the amount of concurrency inherent to a loop body. The example shows a form of "transpose/mapping". The maximal concurrent set of (I, J) points is a triangular helf of the N x N square. In the example, all the generations shown can be performed concurrently (there is no overwrite problem among generators) and then all the uses can be performed concurrently. A subset of these two sets is given by a sequential stepping of a "line" of concurrency parallel to the J axis in the diagram.

```
              DO  1  I = 1, N
              DO  1  J = 1, N
              A(I, J) = ~~~~~~
   1          ~~~ = ~~~A(J, I) ~~~~
```

The DO J loop can be transformed to a DO FOR ALL:

For I = J, intra-loop gen-use precedence,

For I < J, inter-loop generation-use precedence:

Execution at $I_K < J_K$ is generator

for use at $I_L > J_L$

(specifically $I_L = J_K$, $J_L = I_K$)

(Relating K and L:

Let $K = (I - 1) * N + J$

Then $L = (J - 1) * (N - 1) + K$

$= (J - 1) * N + 1$ .)

For $I > J$, inter-loop non-overwrite precedence:

Use at $I_K > J_K$ is of an initial value of the variable.

Generation at $I_L < J_L$ (where $I_L = J_K$, $J_L = I_K$ as before) is for an output or final variable which will not be reused by the loop.

Thus for a given $I_K$,

$$J^* = \{ J_{K-M}, J_{K-M+1} ---, J_{K-1}, J_K, J_{K+1}, --- J_{K+P} \}$$

as an active set for parallel execution gives:

for $J_\alpha < I_K$,

1) generate an output value which won't be used in the loop
2) use a value generated when $I_L = J_\alpha$ (earlier in the I loop)

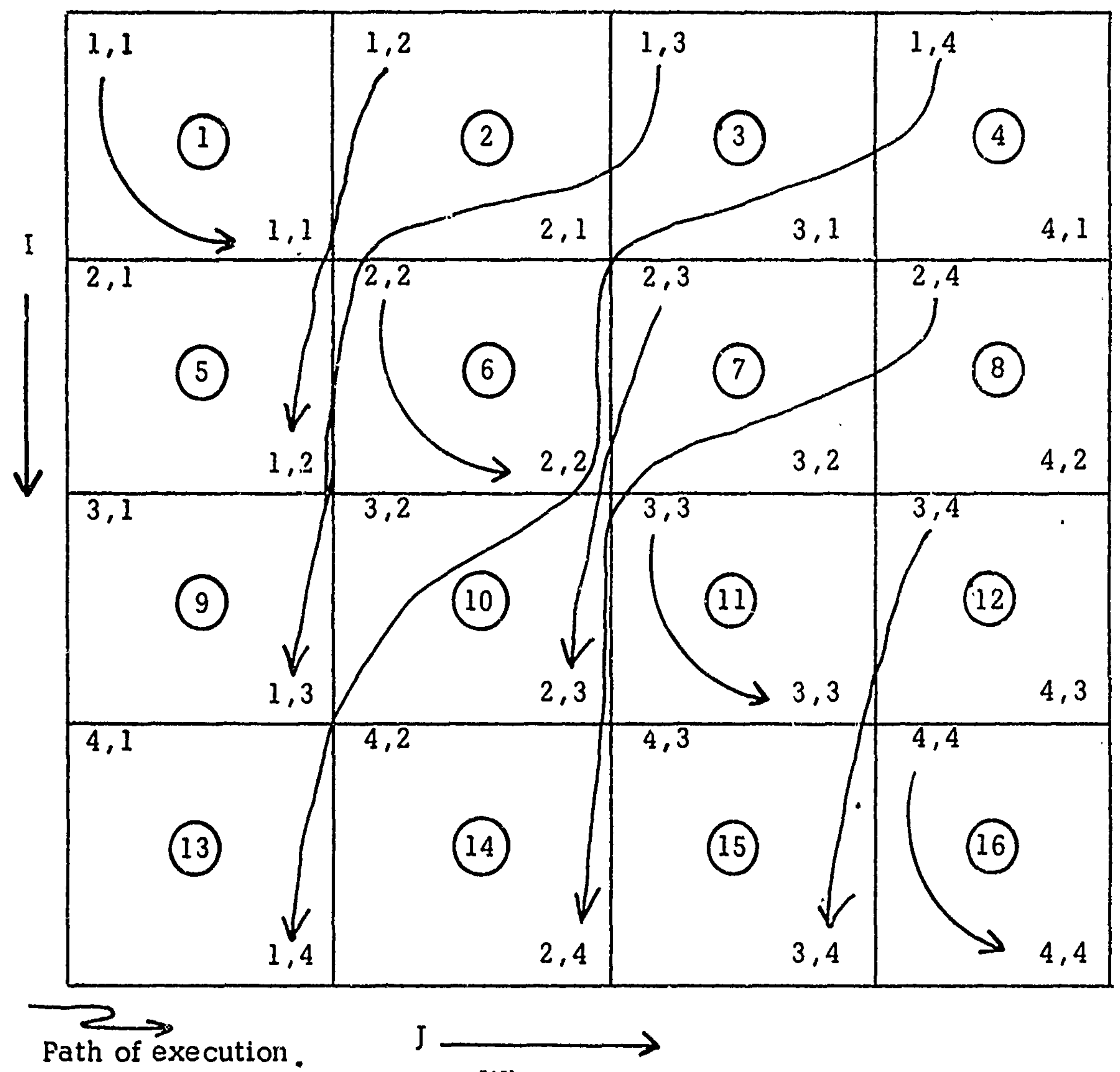
for $J_\alpha = I_K$,

1) generate an output value
2) use the value just generated (intra-loop)

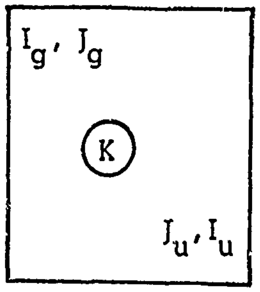
for $J_\alpha > I_K$,

1) generate an output value which will be used in the loop at the point that $I_L = J_\alpha$
2) use an initial value

Diagrammatically, for N = 4, this is:



Key:



Where:

$I_g$, $J_g$ are index pair for A(I,J) = $\sim$ statement

$J_u$, $I_u$ are index pair for = $\sim$ A(J,I) $\sim$ statement

K = 4(I - 1) + J

notes: 1) In the chart $I_g = I_u$, $J_g = J_u$ because what is being modelled is the I, J values resulting for the particular DO statements with their given ordering. (I.e., time is modelled.)

2) All arrows in the diagram point from generation to use.

3) The ordering of the values in the ordered pairs reflects the data point being referenced at a point in time.

It can be seen that all the generations in the "upper right triangular half" of the index set can be executed concurrently, then all the uses. Next the generations and then the uses of the "lower left triangular half" can be executed concurrently. This gives the maximum concurrency and the whole loop can be executed in 2 steps rather than the original 16. By comparison, as has been shown, a DO FOR ALL J rewriting is legal but provides a concurrency improvement factor of only 4 instead of 8. The derivation of generation-use dependency and the sets of index set points for concurrent execution is mechanizable as described in the previous example.

Now we define $J^i_{\sigma+1}$ by

$$\text{(A-2)} \qquad J^i_{\sigma+1} = \begin{cases} J^i_\sigma & \text{if } i \neq p \\ J^p_\sigma + \sum_{r \neq p} q^\sigma_r \quad J^r_\sigma & \text{if } i = p \end{cases}$$

Using (A-2), it is easy to express the $J^i_\sigma$ in terms of the $J^i_{\sigma+1}$. Thus, the mapping $(J^1_\sigma, \ldots, J^n_\sigma) \to (J^1_{\sigma+1}, \ldots, J^n_{\sigma+1})$ is an automorphism of $\mathbb{Z}^n$. Therefore, the mapping $(I^1, \ldots, I^n) \to (J^1_{\sigma+1}, \ldots, J^n_{\sigma+1})$ is also an automorphism of $\mathbb{Z}^n$, because it is the composition of two automorphisms.

Substitution of (A-2) in (A-1) yields

$$\pi[(I^1, \ldots, I^n)] = \Sigma\, \bar{t}_i\, J^i_{\sigma+1},$$

where

$$\bar{t}_i = \begin{cases} t_i - t_p \quad q^\sigma_i & \text{if } i \neq p \\ t_p & \text{if } i = p. \end{cases}$$

It is easy to verify the following two facts:

(i) For each $i \neq p$, if $t_i \neq 0$ then $0 \leqq \bar{t}_i < | t_i |$.

(ii) The g.c.d. at the $\bar{t}_i$ equals the g.c.d. of the $t_i$.

It follows from (i) and (ii) that we will reach a $\sigma \leqq$ minimum $\{ | h_i | : h_i \neq 0\}$ such that in Equation (A-1) we have

$$t_i = \begin{cases} 0 & \text{if } t \neq p \\ 1 & \text{if } i = p \end{cases}$$

for some p. Setting $J^1 = J^p_\sigma$, and $J^2, \ldots, J^n$ equal to the remaining $J^i_\sigma$ then

Now we define $J^i_{\sigma+1}$ by

$$(A\text{-}2) \qquad J^i_{\sigma+1} = \begin{cases} J^i_\sigma & \text{if } i \neq p \\ J^p_\sigma + \sum_{r \neq p} q^\sigma_r \; J^r_\sigma & \text{if } i = p \end{cases}$$

Using (A-2), it is easy to express the $J^i_\sigma$ in terms of the $J^i_{\sigma+1}$. Thus, the mapping $(J^1_\sigma, \ldots, J^n_\sigma) \to (J^1_{\sigma+1}, \ldots, J^n_{\sigma+1})$ is an automorphism of $\mathbb{Z}^n$. Therefore, the mapping $(I^1, \ldots, I^n) \to (J^1_{\sigma+1}, \ldots, J^n_{\sigma+1})$ is also an automorphism of $\mathbb{Z}^n$, because it is the composition of two automorphisms.

Substitution of (A-2) in (A-1) yields

$$\pi[(I^1, \ldots, I^n)] = \Sigma\, \bar{t}_i\, J^i_{\sigma+1},$$

where

$$\bar{t}_i = \begin{cases} t_i - t_p \; q^\sigma_i & \text{if } i \neq p \\ t_p & \text{if } i = p. \end{cases}$$

It is easy to verify the following two facts:

(i) For each $i \neq p$, if $t_i \neq 0$ then $0 \leqq \bar{t}_i < | t_i |$.

(ii) The g.c.d. at the $\bar{t}_i$ equals the g.c.d. of the $t_i$.

It follows from (i) and (ii) that we will reach a $\sigma \leqq$ minimum $\{ | h_i | : h_i \neq 0\}$ such that in Equation (A-1) we have

$$t_i = \begin{cases} 0 & \text{if } t \neq p \\ 1 & \text{if } i = p \end{cases}$$

for some p. Setting $J^1 = J^p_\sigma$, and $J^2, \ldots, J^n$ equal to the remaining $J^i_\sigma$ then

# APPENDIX B

## LINEAR SUBSCRIPT EXPRESSIONS

The algebraic loop analysis methods depend on the subscript expressions within array references, being linearly dependent on the loop index variables. Given:

```
[DO ~~  I ~~~
[N  =  ~~~ I ~~~~
[ ~~~ A(~N~) ~~~~
```

determining if the subscript of A is a linear function of I, involves "back-substituting" for all variables in the subscript. The algorithm is obvious, though painstaking and involves algebra on a canonical form

$$m = ( \Sigma a_i j_i ) + a_o$$

where the $j_i$ are the variables of the DO statements of a nest and $\{a_i\}$, $a_o$ are arbitrary expressions <u>not</u> involving these variables. If any chain of back substitution cannot be put into the canonical form, the subscript is <u>not</u> linear.

A complexity arises from the following example:

```
      N = 0
      DO  10 I = 1, 100
      N = N + 5
      A(I) = B(N) + C(I)
10    CONTINUE
```

which is equivalent, in effect, to:

```
        DO 10 I = 1, 100
        A(I) = B(5 * I) + C(I)
     10 CONTINUE
```

Note that this is the essential truth behind the sequential machine optimization method known as "reduction in operation strength". What we want to do here however is the opposite transformation possibly described as "solution of recursion relations". For recursions of the form $n = n + a$, the solution will be linear in the loop index variable.

Two more general examples are:

(1) $N = N_o$

$$\left[\begin{array}{l} \text{DO} \sim\sim\ I = M_o,\ M_F,\ M_\Delta \\ \wr \\ N = N + K \end{array}\right.$$

$$\Downarrow$$

$$\left[\begin{array}{l} \text{DO} \sim\sim\ I = M_o,\ M_F,\ M_\Delta \\ N = \frac{K}{M_\Delta}\ I + \left(K + N_o - \frac{M_o K}{M_\Delta}\right) = \left(\frac{I - M_o}{M_\Delta} + 1\right) K + N_o \end{array}\right.$$

Note that the end product of the transformation cannot be expressed in <u>integer</u> FORTRAN variables and operations, in general, because the results of the divisions are not likely to yield integers although the final form will yield integer results.

(viz. $M_o = 1,\ N_o = 0,\ K = 3,\ M_\Delta = 2$)

```
(2)   N = $N_o$
      DO ~ I = $M_o^I$ , $M_F^I$ , $M_\Delta^I$
          DO ~ J = $M_o^J$ , $M_F^J$ , $M_\Delta^J$
              N = N + K
```

$$\Downarrow$$

```
      DO ~ I = $M_o^I$ , $M_F^I$ , $M_\Delta^I$
          DO ~ J = $M_o^J$ , $M_F^J$ , $M_\Delta^J$
              $N = \left(\frac{K}{M_\Delta^J}\right) J + \left( \max\left(0, \left\lfloor \frac{M_F^J - M_o^J}{M_\Delta^J} \right\rfloor\right) + 1 \right) \left(\frac{K}{M_\Delta^I}\right) I$
                  $+ \left[ N_o + K - \frac{M_o^J K}{M_\Delta^J} - \left(\frac{M_o^I}{M_\Delta^I}\right) \left( \max\left(0, \left\lfloor \frac{M_F^J - M_o^J}{M_\Delta^J} \right\rfloor\right) + 1 \right) K \right]$
```

where $\lfloor x \rfloor$ = greatest integer less than or equal to x

(note $\left\lfloor \frac{a}{b} \right\rfloor$ for a, b $\epsilon$ N is the FORTRAN integer division operation.)

The messy (max $(0, \lfloor \sim \rfloor) + 1)$ coefficient on I counts the number of iterations of the J-loop performed between I-loop iterations. When $M_o^I = M_o^J = M_\Delta^I = M_\Delta^J = 1$, the above all simplifies to:

$$N = (K)\, J + (M_F^J\ K)\, I + (N_o - M_F^J\ K)$$

The position of the subscript variable "recursive" generation relative to its use as a subscript can have an effect on the form of the linear expression substituted into the subscripts:

a

```
$N = N_o$
┌ DO ~ I ~
│ $N = N + K$
└ ~ A(N) ~

        ⇓

┌ DO ~ I ~
└ ~ A($K * I + N_o$) ~
```

b

```
$N = N_o$
┌ DO ~ I ~
│ ~ A(N) ~
└ $N = N + K$

        ⇓

┌ DO ~ I ~
└ ~ A($K * I + N_o - K$) ~
```

## BIBLIOGRAPHY

[1] McIntyre, David: An Introduction to the ILLIAC IV Computer. Datamation, Vol. 16, No. 4 (April, 1970).

[2] Muroaka, Yoishi: Parallelism Exposure and Exploitation in Programs. Ph.D. Thesis, University of Illinois, Urbana, 1971.

[3] Mordell, L.J.: Diophantine Equations. Academic Press, N.Y. (1969).

[4] Myszewski, Mathew: IVTRAN: A Dialect of FORTRAN for Use on the ILLIAC IV. (In preparation) Applied Data Research, Inc., Wakefield, Massachusetts.

[5] Warshall, Stephen: A Theorem on Boolean Matrices, Journal of the ACM, Vol. 9, (1962), pp. 11-12.

[6] Ramamoorthy, C.V. and Gonzaleg, M.J.: "A Survey of Techniques for Recognizing Parallel Processable Streams in Computer Programs", in Proceedings AFIPS Fall Joint Computer Conference, 1969, pp. 1-15.